TEXAS INSTRUMENTS

Improving Man's Effectiveness Through Electronics

Model 990 Computer
TX990 Operating System
Programmer's Guide
(Release 2)

MANUAL NO. 946259-9701
ORIGINAL ISSUE 1 APRIL 1977
REVISED 15 DECEMBER 1977

Digital Systems Division

LIST OF EFFECTIVE PAGES

INSERT LATEST CHANGED PAGES DESTROY SUPERSEDED PAGES

Note: The portion of the text affected by the changes is indicated by a vertical bar in the outer margins of the page.

Model 990 Computer TX990 Operating System Programmer's Guide
(Release 2) (946259-9701)

Original Issue . . . . . 1 April 1977
Revised . . . . . 15 December 1977 (ECN 41915)

Total number of pages in this publication is 274 consisting of the following:

PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.
Cover	0	Appendix A Div	0	Appendix I Div	0
Effective Pages	0	A-1 - A-4	0	I-1 - I-2	0
iii - xiv	0	Appendix B Div	0	Appendix J Div	0
1-1 - 1-8	0	B-1 - B-16	0	J-1 - J-2	0
2-1 - 2-6	0	Appendix C Div	0	Appendix K Div	0
3-1 - 3-22	0	C-1 - C-8	0	K-1 - K-2	0
4-1 - 4-10	0	Appendix D Div	0	Appendix L Div	0
5-1 - 5-6	0	D-1 - D-4	0	L-1 - L-6	0
6-1 - 6-14	0	Appendix E Div	0	Appendix M Div	0
7-1 - 7-26	0	E-1 - E-2	0	M-1 - M-4	0
8-1 - 8-16	0	Appendix F Div	0	Alphabetical Index Div.	0
9-1 - 9-38	0	F-1 - F-2	0	Index-1 - Index-6	0
10-1 - 10-6	0	Appendix G Div	0	User's Response	0
11-1 - 11-6	0	G-1 - G-2	0	Business Reply	0
12-1 - 12-2	0	Appendix H Div	0	Cover Blank	0
13-1 - 13-6	0	H-1 - H-2	0	Cover	0

PREFACE

This manual enables the user to employ the TX990 Operating System, in its standardized or customized modular configuration (including associated utility programs), with the user's standardized or customized hardware configuration.

The sections and appendixes of this manual are organized as follows:

I General Description – Provides a general description of the TX990 Operating System and its utility programs.

II Loading the Operating System – Provides several alternative step-by-step procedures for loading the Operating System.

III Operator Communication Package (OCP) – Describes the keyboard commands available to the operator for communicating interactively with the TX990 Operating System (e.g., to load a program, abort a task, abort an I/O operation, debug a program, set and clear breakpoints, et al).

IV Control Program – Describes how to use the Control Program to load and execute program from the console.

V Programming Tasks – Describes how to program user tasks to run under TX990.

VI Executive Supervisor Calls – Describes how the user can employ the executive management capability of the Operating System through programmed supervisor calls.

VII Device and File I/O Supervisor Calls – Describes how the user can employ the executive management capability of the Operating System for controlling an input/output device or file.

VIII Diskette OCP System Utility (SYSUTL) Program – Describes the keyboard commands available to the operator for communicating interactively with diskette devices and files.

IX System Generation (GENTX) Utility Program – Describes how the user can customize an operating system for a specific hardware and software configuration.

X Diskette/Disc Backup and Initialize (BACKUP) Utility Program – Describes how to copy (or backup) and verify diskettes, as well as initialize new diskettes.

XI Object Manager (OBJMGR) Utility Program – Describes how to copy standard and compressed object modules from diskette-, cassette-, or card-files onto diskette or cassette files and how to organize the files by deleting or adding modules.

XII LIST80/80 (LIST 80) Utility Program – Describes how to copy 80-character records from one device or file to another.

XIII Diskette Dump (DSKDMP) Utility Program – Describes how the user can load, display, and modify diskettes on an allocation unit basis.

A List of Supervisor Calls – Provides a list of all the TX990 Operating System supervisor calls.

B Device Character Sets – Lists the character sets for the 911 VDT, 913 VDT, 733 or 743 Data Terminal, 804 Card Reader, 306, 2230, 2260 and 588 Line Printers, 33 ASR Teletypewriter.

C User-Supplied Modules – Describes user-supplied software modules that may be required for unique peripheral devices, user's extended operations, or user supervisor calls.

D Glossary – Clarifies selected words used in this TX990 Operating System Programmer's Guide.

E TX990-DX10 Compatibility – Describes the considerations to be met to achieve upward compatibility of tasks from the TX990 Operating System to the DX10 Operating System.

F Compressed Object Code Format – Describes the compressed object code format.

G Task State Codes – Lists and describes the task state codes.

H Printout of Fatal Task Error Codes or Display of Illegal Interrupt Code – Lists and describes the fatal task error codes and the illegal interrupt code.

I I/O Error Codes – Lists and describes the I/O error codes available to the user, when coding a program, for printout or display on a terminal device.

J System Tasks – Describes the eight system tasks capable of being included in the TX990 Operating System.

K System Generation using DX10 Release 3.0 – Steps involved in TX990 system generation on a DX10 system.

L Support for the 32 I/O Module – Describes the I/O supervisor call and system generation support for the 32 I/O module supported by TX990 as a special device.

M Support for the 5MT/6MT Module – Describes the I/O supervisor call and system generation support for the 5MT/6MT special device supported by TX990.

The following documents contain additional information related to the TX990 Operating System and are referenced herein this manual:

Title	Part Number
Model 990 Computer Terminal Executive Development System (TXDS) Programmer's Guide	946258-9701
Model 990 Computer TMS9900 Microprocessor Assembly Language Programmer's Guide	943441-9701
Model 990 Computer Model FD800 Floppy Disc System Installation and Operation	945253-9701

Title	Part Number
Model 990 Computer Model 913 CRT Display Terminal Installation and Operation	943457-9701
Model 990 Computer Model 911 Video Display Terminal Installation and Operation	945423-9701
Model 990 Computer Model 733 ASR/KSR Data Terminal Installation and Operation	945259-9701
Model 990 Computer Model 804 Card Reader Installation and Operation	945262-9701
Model 990 Computer Models 306 and 588 Line Printers Installation and Operation	945261-9701
Model 990 Computer PROM Programming Module Installation and Operation	945258-9701
990 Computer Family Systems Handbook	945250-9701
Model 990 Computer Communications Systems Installation and Operation	945409-9701
Model 990 Computer DX10 Operating System Programmer's Guide	945257-9701
Model 990 Computer Communications System Software	946236-9701

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

APPENDIXES

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION I

GENERAL DESCRIPTION

1.1 OVERVIEW

The TX990 operating system is an executive that controls task execution in a real time environment. TX990 executes in either the Model 990/4 or the Model 990/10 Computer. The memory protect hardware option (Model 990/4) and memory mapping option (Model 990/10) are not used by TX990. The maximum memory size is 56K bytes in the Model 990/4 Computer and 62K bytes in the Model 990/10 Computer.

TX990 may be customized to provide many features for the user, and configured to save memory by excluding features which are not desired. A minimum TX990 system includes a task scheduler, interrupt handler for the clock, and a supervisor call interface, and occupies about 4K bytes of memory. Additional modules and features (e.g., file management, operator communications package) are optional according to application requirements.

The task scheduler provides multiple priority level scheduling of Central Processor Unit (CPU) time and maintains a list of active tasks at each priority level. Four levels of priority are available for user tasks. The TX990 Operating System allocates time to tasks by time slicing. Tasks are allocated time slices at each of the four priority levels on a first-in, first-out (FIFO) basis. During system generation the user may specify the maximum number of time slices that may occur at each priority level before giving the next priority level a time slice. The length of time slice is also specified during system generation.

The interrupt handling software operates in conjunction with the prioritized interrupt scheme to allow the user to assign high interrupt priorities to critical devices or input lines. The interrupt handler must be included in the operating system to support clock-interrupt time slicing and time-dependent supervisor calls.

Tasks request the support of the operating system by executing supervisor calls (SVCs). SVCs are routed through a supervisor call interface to an associated supervisor call processor. A substantial number of file management supervisor calls and task control calls are supported by the operating system. The user may also include his own custom supervisor call processors if desired.

Tasks that execute under TX990 may be classified as either dynamic tasks or tasks linked into the operating system. When a TX990 operating system is generated, its various parts must be link edited to form a single linked object module, which can be loaded into memory and executed as an operating system. Any system or user tasks linked with the other system modules are loaded into memory when the system is booted, and are memory resident.

Other tasks, called dynamic tasks, that are not linked with the system may reside in diskette files or cassettes. They may be dynamically loaded and executed through the use of operator commands or supervisor calls. All user tasks not linked in with the system are dynamic tasks. Dynamic tasks are loaded beyond the end of the operating system and all linked-in tasks, into memory called the dynamic task area. The TX990 operating system may be customized to support a single dynamic task, multiple dynamic tasks, or no dynamic tasks.

1.2 SUPPORTED HARDWARE

The Model 990 Computer System hardware supported by TX990 include the following:

- Model 990/4 or 990/10 Computer
- Programmer Panel
- FD800 Floppy Disk
- Model 911 or 913 Video Display Terminal
- Model 733 "Silent 700"* ASR/KSR Data Terminal
- Model 743 Data Terminal
- Model 306 Line Printer
- Model 588 Line Printer
- Model 810 Line Printer
- Model 2230 Line Printer
- Model 2260 Line Printer
- Model 804 Card Reader
- 33 ASR Teletype Data Terminal
- 5MT/6MT Serial Interface Module
- 32-In/Transition Detection Module

1.3 FILE MANAGEMENT FEATURES

The TX990 operating system optionally provides a file management package to support file structures and operations on diskettes and file-oriented devices. File management maintains a directory on all initialized diskettes in the system (see Section X on diskette initialization). This directory can point to up to 48 files on the diskette. Diskette files consist of blocks of allocation units. An allocation unit is a logical division of the diskette. A diskette is divided into 333 allocation units, each consisting of six sectors. Two basic file types are supported: sequential files and relative record files. Additionally, a special usage of the relative record file, called a program file, is supported. Some of the features of the file management package and file types are described in the following paragraphs.

1.3.1 VOLUME NAMES. In addition to accessing files on a particular diskette drive (e.g., DSC, DSC2), file management optionally supports volume names for initialized diskettes. When a diskette is initialized, the user may assign it a one to four character volume name. If volume name support is included in the operating system, files on that diskette may be accessed through the volume name or the diskette drive device name. Addressing a diskette by volume name can be performed regardless of which drive it may be mounted on.

*Trademark of Texas Instruments Incorporated

Volume names are only supported by operating systems which include the module VOLUME (see the section on system generation). The standard TI supplied TX990 operating systems do not provide volume name support and diskettes may only be accessed by drive name.

1.3.2 **SEQUENTIAL FILES.** The logical data records in a sequential record file must be accessed in a sequential manner (i.e., record 1 must be processed before record 2, etc.). When a sequential file is closed following an access, the operating system saves the position of the last access to the file. When the file is opened again, the next I/O call accesses the next logical record in the file instead of the first logical record of the file. To access the first logical record, the file must be rewound before the access.

The last write operation performed on a file defines the current end of that file. No data can be read from the media that is beyond the current end of that file. The operating system prevents switching from a read, backspace or forward space operation to a write or write end-of-file operation until an end-of-file mark is read; however, when extending a file, a backspace of one record can be followed by write operations. This precaution prevents writing on data previously stored in a file. Similarly, the operating system prevents switching from a write operation to a read, forward space, backspace, or rewind operation until an end-of-file mark is written.

Sequential files are blank compressed: i.e., when a logical record is written to a file, ASCII blanks are removed from the record. Blanks are restored to the record as it is read from the file.

1.3.2.1 **Sequential Cassette File.** A cassette may contain a single file that fills one side of the cassette, or it may contain multiple subfiles with an end-of-file mark separating each individual file. The beginning of the first file on the tape is located by executing a rewind operation. Subsequent subfiles can be located by forward spacing or reading to the next end-of-file mark. Any number of subfiles may appear on the cassette; however, the length of the file is limited to one side of the cassette.

1.3.2.2 **Sequential Diskette File.** A sequential diskette file is comprised of logical records in blocks of allocation units. Whenever possible, the operating system will allocate contiguous blocks of allocation units. Diskette files may be created by either the Create File operation (code 90_{16}), or through the autocreate function when the file is opened. The operating system allocates additional allocation units to the file as needed when writing to the file. Each diskette file can contain a maximum of 20 noncontiguous blocks. Each file is terminated by an end-of-file mark. The beginning of the file can be located by executing a rewind file operation. Subfiles can be created by inserting end-of-file marks within the file. To add a new record to the end of a existing file or last subfile, read (or forward space) until the end-of-file mark is detected. Then backspace one record to position the media on the end-of-file record and write the new record(s).

1.3.3 **RELATIVE RECORD FILES.** Relative record files are supported only on diskette. A relative record file is comprised of contiguous and noncontiguous blocks of logical records, in or out of sequence (i.e., in random sequence). Each logical record is identified by a logical record number. Space is automatically allocated for a relative record file in blocks of contiguous allocation units when possible and, when not possible, in blocks of noncontiguous allocation units. There is no maximum number of logical records which can be placed in the file. However, there can be no more than 20 noncontiguous blocks of allocation units. The beginning of a relative record file is specified by logical record 0; the end of the relative record file is specified by the highest numbered logical record, which is automatically identified with an end-of-file mark. No subfiles exist in relative record files.

Logical records in a relative record file have a fixed length defined when the file is created. If the user tries to write a record larger than the specified record length, the record is truncated. If the record is shorter than the fixed record length, the record is binary zero filled.

By initially coding the logical record number to zero, the relative record file may be automatically written or read sequentially until the highest logical record number is reached. After a read or write I/O file operation, the logical record number is incremented by the file management task; the logical record number is also incremented, or decremented, when the relative record file is rewound, forward spaced or backward spaced. The user may access any record at random by specifying the desired logical record number. Subsequent accesses may be either random or sequential. When another relative record number is specified out of sequence, access is at random. When none is specified, the next record in sequence is accessed.

1.3.4 PROGRAM FILES. Program files are relative record files used to contain a program memory image; i.e., the linked object of one task segment and possibly one procedure segment and up to 256 overlay segments. Program files are created by the Link Editor in IMAGE format. A program file may contain only one task and one procedure.

A program file contains several records of system overhead information, and then the linked object of a program (task, procedure, and overlays). The object code is in a format such that it can be loaded directly into memory by the image loader. By linking programs into program files, the user may save diskette space and load time. Programs in program files take 25%-40% less time to load than programs in compressed or noncompressed object files, and require approximately 25% less diskette space.

1.4 LOGICAL I/O

TX990 uses logical unit numbers (LUNOs) to represent physical devices and files. User tasks are coded to perform I/O to a LUNO, which is converted by the operating system to represent the physical device or file to which the LUNO is assigned. This enables a user to code an I/O operation independent of a specific hardware or file configuration.

LUNOs may be assigned and released by a task using I/O Supervisor Call 00 operations or operator commands. A LUNO is assigned to a pathname, which may be either a device name (1-4 characters) or a file name. Pathname format and syntax is described in the following paragraph. LUNOs may range in value from 0 to FF_{16}.

1.4.1 PATHNAMES. A pathname may be the name of a device or a file. If a pathname represents a device, it is the name (1-4 characters, first must be alphabetic) assigned to that device at system generation (e.g., CS1, LP, CR).

If a pathname represents a file, it has the following format:

$$\left[\left\{\begin{array}{l}\text{<device name>}\\ \text{<volume name>}\end{array}\right\}\right] \text{:<file name>} \left[\text{/ [<extension of file name>]}\right]$$

The first pathname component is optional, and is the name of the diskette drive or diskette volume which contains the file. A volume name is 1-4 characters, like a device name. The first character must be alphabetic; the remaining characters must be alphanumeric. A volume name may only be used if volume name support is selected during system generation (see Section IX). TI supplied TX990 systems do not include volume name support. The default value for this pathname component is defined during system generation.

The second pathname component consists of a colon (:) and a 1-7 character file name. The first character of the file name must be alphabetic; the rest may be alphanumeric.

The third pathname component is an optional 1-3 character extension of the file name. The first character must be alphabetic or blank; the rest must be alphanumeric or blank. If an extension is used, it must be separated from the file name by a slash (/) character. Optionally, the slash may be used alone, and the extension is interpreted to be three blanks. If no slash-extension is used, the extension is again interpreted as blanks.

The following are examples of legal pathnames:

:FILE	File name, 1 to 7 characters; the device name is the default disk name.
DEV:FILE	Device and file name; the extension is defined as blanks.
VOL:FILE	Volume Name 1 to 4 characters; the extension is defined as blanks.
:FILE/EXT	File name and file extension name; the extension can be 1-3 chars. The device name is the default disk name.
DEV:FILE/EXT	Complete pathname with no defaults.
VOL:FILE/EXT	Complete pathname with no default
DEV:FILE/	Defaults to a blank extension
VOL:FILE/	Volume name, and file name, extension defaults to blank extension.

Trailing blanks are allowed at the end of each field (as presented below), but embedded blanks are not allowed within the field itself.

DSC:FILEbbb/bbb is the same as :FILE/or :FILE

NOTE

The above example assumes that DSC was generated as the default diskette name during system generation.

1.4.2 I/O MODES. Three I/O modes of operation are employed: file mode, record mode, and character mode. I/O devices are designated for file, record or character mode of operation during system generation. An I/O device placed in the file mode is assigned to the calling task when an open I/O operation is employed and remains assigned to that calling task until a close operation is performed. An I/O device placed in the record mode remains assigned to the calling task only during execution of the individual I/O operation. In the record mode, a Video Display Terminal (VDT) has a LUNO assigned to enable execution of an I/O operation. However, a 911 or 913 VDT is also capable of being operated in the character mode. The character mode enables the VDT to be uniquely programmed and accessed by a station number, rather than a LUNO.

1.5 SUPERVISOR CALLS

Tasks request the support of the operating system by executing supervisor calls that are routed through a supervisor call interface to an associated supervisor call processor. A substantial number of file management supervisor calls and task control calls are supported by the operating system. The user may also include his own custom supervisor call processors if desired. The different supervisor calls provided by TX990 are described in Sections VI and VII.

1.6 OPERATOR INTERFACES

TX990 optionally provides two operator interfaces, the Operator Communications Package (OCP) and the Control Program. A customized TX990 system may include neither, one, or both of the interface packages. The Control Program prompts the operator to load and execute any system utility or user program. It also provides a means of passing parameters to the program. The Operator Communications Package provides several commands which may be entered by the operator, and which perform various system functions (e.g., loading or executing programs, and debugging capabilities). The two operator interfaces are described in Sections III and IV.

1.7 SYSTEM MEMORY LAYOUT CONSIDERATIONS

Figure 1-1 shows the flow of control within a TX990 that includes all options. The task scheduler initiates execution of a user task as previously described. The task requests support of the executive by executing SVCs that are processed by the supervisor call interface. I/O supervisor calls are processed by the I/O Supervisor (IOS), and other supervisor calls are processed by their respective supervisor call processors. Following execution of a supervisor call, the supervisor call interface returns control either to the calling task or to the task scheduler, according to the type of supervisor call. Interrupts from the clock and from active I/O devices are processed by the appropriate handler, which passes control to a common interrupt return after processing the interrupt. The common interrupt return passes control to the point in the task or executive at which the interrupt occurred.

In figure 1-1, the optional portions of TX990 are shown outlined with dashed lines. Figure 1-2 shows memory maps of several possible configurations of TX990. The minimal system consists of the basic configuration shown in figure 1-2A. This provides a basic task scheduling and an interrupt processing monitor for a small static system. The minimal TX990 Operating System includes the data structures, task scheduler, supervisor call interface (with dummy supervisor call processors), and the interrupt processor. The user tasks in this type of system are linked and loaded with the system and perform any I/O operations without the aid of TX990.

The addition of supervisor call processor modules, I/O supervisor call processor modules, and Device Service Routine (DSR) modules results in the configuration shown in figure 1-2B. In this configuration, it is necessary to link and load user tasks with the system.

The addition of OCP modules as shown in figure 1-2C adds the capability of operator interaction with the oSystem ssing OCP commands. Addition of a dynamic task area, also shown in figure 1-2C, allows the operator to load a task or tasks by OCP command into the dynamic task area for execution. This capability, along with the use of OCP commands, enables debugging of tasks. Other user tasks must be linked and loaded with the operating system.

The addition of file management modules, as shown in figure 1-2D, adds the capability of performing I/O operations with diskettes. This capability is necessary when the user wants to configure the Operating System using Terminal Executive Development System (TXDS). TXDS provides the user with those utilities necessary for software development (see the *TXDS Programmer's Guide*, part number 946258-9701, for details).

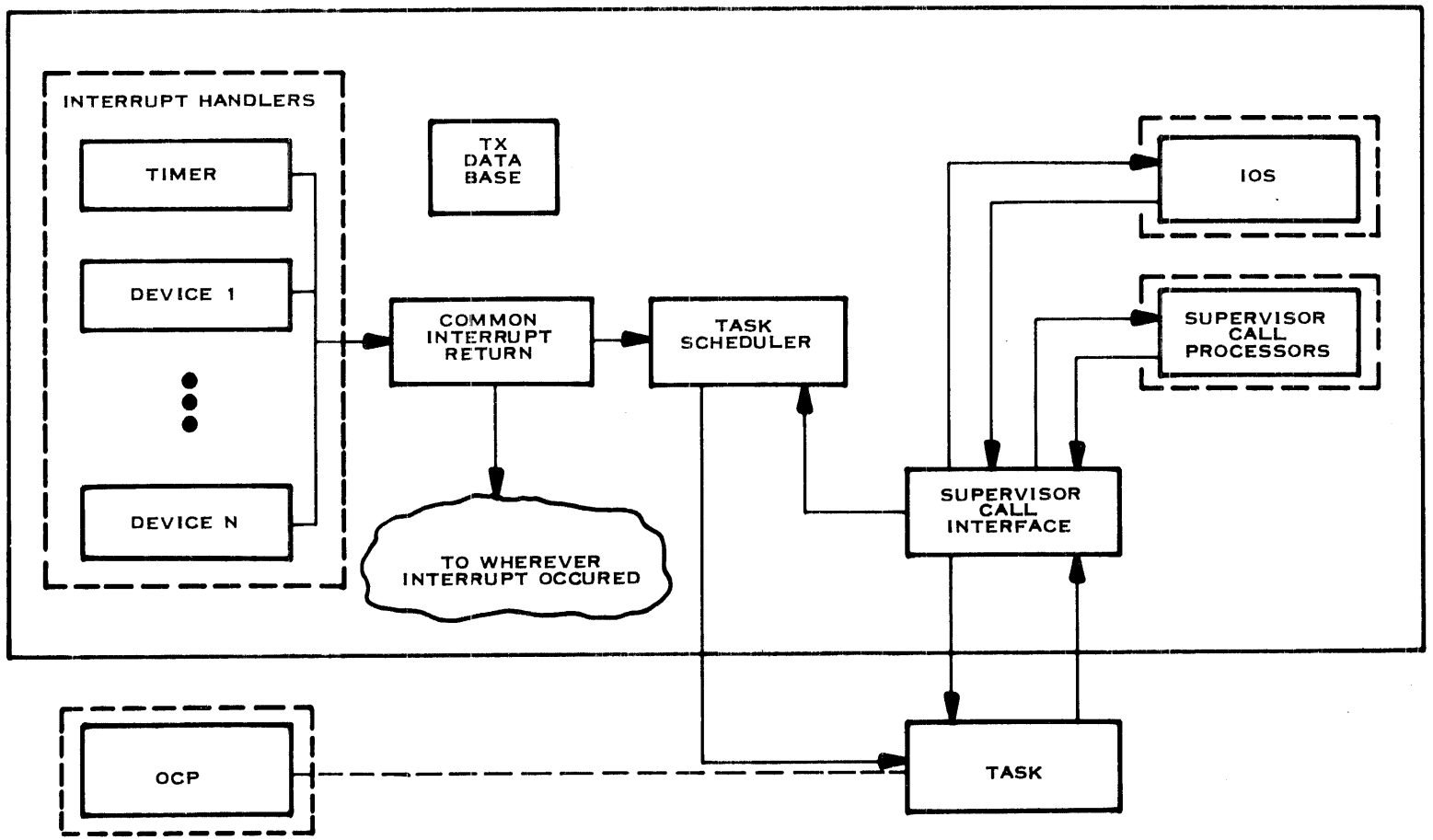


Figure 1-1. TX990 Control Flow

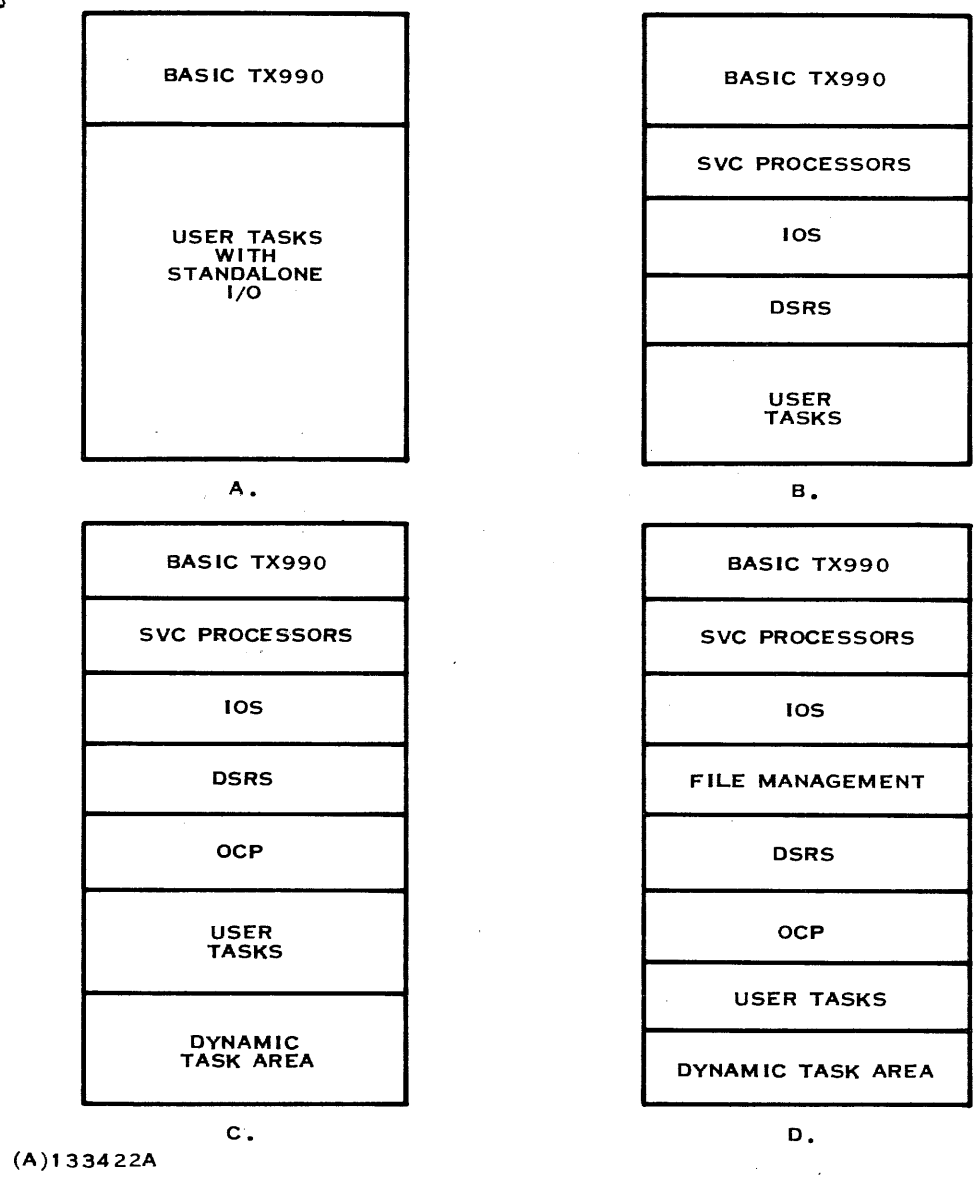


Figure 1-2. Typical TX990 Operating System Configurations

A system configured as shown in figure 1-2A requires less of available memory for system purposes (overhead) than the other configurations, and is desirable where minimizing memory overhead is the most important consideration. The additional support provided by the other configurations requires that more memory be available for system use.

Systems configured as shown in figure 1-2A and B support user tasks that are identified to the operating system during system generation and are linked and loaded with the operating system. These tasks occupy the area of memory beyond the area occupied by TX990, and may be placed in execution when the operating system is loaded. Operating Systems configured as shown in figure 1-2C and 1-2D also support these types of tasks but, in addition, they support a dynamic task area into which tasks may be loaded and executed by OCP command. The dynamic task area consists of the available memory beyond the area occupied by the other user tasks.

SECTION II

LOADING THE OPERATING SYSTEM

2.1 INTRODUCTION

The TX990 operating system may be loaded into memory from a diskette or a cassette. Basically, two methods may be used to load the operating system: (1) a ROM loader can execute and load the TXBOOT program from diskette which in turn loads the Operating System; or (2) a ROM loader can execute and load the operating system directly into memory from a cassette or cards. More specifically, when the operating system is loaded from a diskette, a diskette/cassette ROM loader is utilized and the TXBOOT loader is loaded from a diskette; when the operating system is loaded from cassette, a diskette/cassette ROM loader or a card/cassette ROM loader is utilized.

This section provides the step-by-step procedures required to achieve successful loading with each method. The initilization procedure required to be performed after the operating system is loaded into memory is also presented in this section. Two alternative initialization procedures are presented: one is used when an Operator Communication Package (OCP) is supplied with the operating system and the other is used when the TX990 Control Program and no OCP is supplied with the operating system. Any operator interaction with an operating system that does not include OCP capability must be provided by either user tasks or user-supplied system modules. At the end of this section, a procedure is provided to manually restart the system in event a system failure does not cause the contents of memory to be altered.

When an error occurs, the error code is displayed by the indicators on the front of the Programmer Panel and the FAULT indicator is turned on. (See Appendix H, Printout of Fatal Task Error Codes or Display of Illegal Interrupt Code.) When an Operator Panel is used in the system, any error which occurs during loading causes the FAULT indicator to turn on and stay on.

2.2 LOADING THE TX990 OPERATING SYSTEM

The following paragraphs describe four methods for loading a TX990 operating system.

2.2.1 LOADING FROM DISKETTE. Proceed as follows:

1. Insert one of the three operating system diskettes (i.e., the diskette which contains the 913 VDT operating system, the 733 ASR operating system, or the 911 VDT operating system) into any diskette drive.

2. Sequentially depress the HALT/SIE, RESET, and LOAD switches on the front of the programmer panel. The TXBOOT program will now load into memory, followed by the TX990 operating system.

NOTE

The ROM loader tries to load from each of the first four diskette drives, beginning with diskette drive 0, until a READY drive is found. If none of the diskette drives indicate READY, the ROM loader will load from the cassette drive that is in the PLAYBACK mode and in the READY state.

3. When successful loading of the TXBOOT program and the operating system is completed, observe the following printed output or display from the system console (provided the Initial Start Task (STASK) module-feature is supplied with TX990):

   TX990 SYSTEM RELEASE 2.2

   MEMORY SIZE (WORDS): 16384 AVAILABLE: 5940

   (where the memory size printout or display is in decimal numbers specifying the quantity of memory words; the size, in words, of the dynamic task area is printed out as available memory).

NOTE

1. When the user does not include a task that produces an indication of successful loading of the operating system, the only possible way to verify successful loading is by usage of the operating system.

2. The operating system diskette being used during the loading procedure must have the diskette boot program written on allocation units 0 through 4, and the system file pathname must be defined. All standard TI diskettes have the diskette boot and the system file pathname previously defined; however, if the user wants to create an operating system diskette, the diskette boot program can be written and the system file pathname can be defined by using the Set System File (SF) command from the Diskette OCP System Utility (SYSUTL) Program. For information relating to SYSUTL, see the Diskette OCP System Utility (SYSUTL) Program section in this manual.

4. After loading of the operating system is completed (i.e., after the printed output or display appears on the system console as specified in the previous step), perform the initialization procedure presented in paragraph 2.3.

2.2.2 **LOADING FROM CASSETTE USING DISKETTE/CASSETTE ROM LOADER.** Proceed as follows:

1. Place the cassette which contains the TX990 operating system in either of the two cassette drives of the 733 ASR terminal.

NOTE

For operating instructions covering the cassette unit, refer to the *Model 990 Computer 733 ASR/KSR Data Terminal Installation and Operation Manual*, part number 945259-9701.

2. Rewind the operating system cassette.

3. Place the cassette unit in the PLAYBACK mode and in the READY state.

4. Press the HALT/SIE switch and then the RESET switch on the Programmer Panel.

5. Place the number 0080_{16} into memory address 0080_{16} as follows:

   a. Press the CLR switch and the 8 switch on the Programmer Panel to set the displayed value to 0080_{16} .

   b. Press the ENTER MA switch on the Programmer Panel.

   c. Press the MDE switch on the Programmer Panel.

6. Load TX990 into memory by pressing the LOAD switch on the Programmer Panel.

7. When loading of the operating system program is completd, observe the following printed output or display from the system console (provided the Initial Start Task (STASK) module-feature is supplied with TX990):

   TX990 SYSTEM RELEASE 2.2

   MEMORY SIZE (WORDS): 16384 AVAILABLE: 5940

   (where the memory size printout is in decimal numbers specifying the quantity of memory words; the size, in words, of the dynamic task area is printed as available memory).

8. After loading of the operating system is completed (i.e., after the printed output or display appears on the system console as specified in the previous step), perform the initialization procedure presented in paragraph 2.3.

2.2.3 **LOADING FROM CASSETTE USING CARD/CASSETTE ROM LOADER.** Proceed as follows:

1. Place the TX990 operating system cassette in either of the two cassette drives of the 733 ASR terminal.

   NOTE

   For operating instructions covering the cassette unit, refer to the *Model 990 Computer Model 733 ASR/KSR Data Terminal Installation and Operation Manual*, part number 945259-9701.

2. Rewind the operating system cassette.

3. Place the cassette unit in the PLAYBACK mode and in the READY state.

4. If the system has an Operator Panel instead of a Programmer Panel, proceed to the next step. If the system has a Programmer Panel, press the HALT/SIE switch and then the RESET switch.

5. Load the operating system into memory by pressing the LOAD switch on the Programmer Panel.

6. When loading of TX990 is completed, observe the following printed output or display from the system console (provided the Initial Start Task (STASK) module-feature is supplied with TX990):

   TX990 SYSTEM RELEASE 2.2

   MEMORY SIZE (WORDS): 16384 AVAILABLE: 5940

   (where the memory size printout is in decimal numbers specifying the quantity of memory words; the size, in words, of the dynamic task area is printed as available memory).

7. After loading of the operating system is completed (i.e., after the printed output or display appears on the system console as specified in the previous step), perform the initialization procedure presented in paragraph 2.3.

2.3 INITIALIZATION

Two initialization procedures are available: one for use when OCP is included in the operating system and the other for use when the Control Program is included. When OCP is included in the operating system, OCP commands may be entered on a system console such as a 733 ASR Data Terminal, or a 913 or 911 Video Display Terminal. The system console is defined during the system generation, and LUNO 0 is assigned to the system console device at that time. Refer to table 2-1 for a list of the standard TX990 device names.

Table 2-1. Standard TX990 Device Names

Device	Device Name
733 ASR/KSR Keyboard/Printer	LOG, ASR
743 KSR Keyboard/Printer	LOG, KSR
733 ASR Cassette Unit 1 (Unit on left)	CS1
733 ASR Cassette Unit 2 (Unit on right)	CS2
Line Printer	LP
Card Reader	CR
913 or 911 Video Display Terminal	CRT, LOG
Dummy	DUMY
Diskette Unit 1 (Unit on left)	DSC
Diskette Unit 2 (Unit on right)	DSC2
Communication Device	COM
Teletype Keyboard/Printer[1]	LOG, ASR
Teletype Paper Tape Reader[1]	PTR
Teletype Paper Tape Punch[1]	PTP

[1] Supported by Texas Instruments as a nonstandard item.

2.3.1 INITIALIZATION USING OCP. Proceed as follows:

1. Enter an exclamation point (!) at the keyboard of the system console. This activates OCP, which responds by printing a period (.) to request entry of a command.

2. Assign LUNO 1 to a printing or display device on which information is to be displayed or printed. Table 2-1 lists device names in the operating system supplied by Texas Instruments. The standard TX990 assigns LUNO 1 to the system console. The following is an example of a command to assign LUNO 1 to a line printer. The command is followed by a carriage return.

   .AL,1,LP.

3. When TX990 includes date and time support, initialize the date and time. The following command initializes the time and date to 3:42 P.M., February 3, 1976. The command is followed by a carriage return.

   .ID,1976,2,3,15,42.

   The operating system responds by printing time and date information (updated approximately), in the following format:

   15:42:18 FEB 3, 1976

At this point, other commands may be entered to perform available functions. The OCP commands are described in Section III.

2.3.2 INITIALIZATION USING THE CONTROL PROGRAM. Proceed as follows:

1. Enter an exclamation point (!) at the keyboard of the system console. This activates the Control Program, which prints the following heading and prompt:

   TXDS 936215 *A

   PROGRAM:

2. Insert the TX990 parts diskette in drive 1.

3. Activate the OCP System Utility Program (SYSUTL) by responding to the PROGRAM: prompt as follows:

   PROGRAM: :SYSUTL/SYS*

4. Initialize the time and date by responding to the SYSUTL OP: prompt as follows:

   TX990 SYSTEMS UTILITY 937544 *B

   OP:ID, <year>, <month>, <day>, <hour>, <minute>. TE.

   SYSUTL returns controls to the Control Program.

2.4 MANUAL SYSTEM RESTART

When the TX990 operating system is used in a Model 990 Computer equipped with a Programmer Panel, it is possible to restart the system without performing another loading procedure. A restart is appropriate in the event a system failure did not alter the contents of the system area of memory. To restart the system, perform the following steps at the Programmer Panel:

1. Transfer control to the Programmer Panel by pressing the HALT/SIE switch.
2. Press the RESET switch.
3. Press the CLR switch and observe that the indicators of the display are not lit.
4. Press the 8 switch and the 10 switch, setting a number of $00A0_{16}$ in the display.
5. Press the ENTER PC switch.
6. Press the RUN Switch.

NOTE

A standard TI-supplied TX990 Operating System does not execute Initial Start Task (STASK) after a restart operation and, therefore, no display or printout is presented on the system console.

7. Perform the initialization procedure presented in paragraphs 2.3.1 or 2.3.2.

SECTION III

OPERATOR COMMUNICATION PACKAGE (OCP)

3.1 INTRODUCTION

OCP allows the user to interactively request actions from the operating system through a terminal (e.g., 733 ASR, 743 KSR, 33 Teletype, 911 or 913 VDT). Actions are initiated by the user by responding to the OCP command prompt (a period) with one of the OCP commands described in paragraph 3.5.

Processing of operator commands is performed by five or more modules of the Operator Communication Package (OCP). OCP consists of four required modules and five optional command processor modules. These modules are described in the system generation section of this manual. The user may modify the OCP commands, add one or more command processors to a command processor module, or add one or more command processor modules to OCP.

3.2 ACTIVATING AND DEACTIVATING OCP

In a TX990 system which includes OCP, it is initiated at the system console by entering an exclamation mark (!). OCP responds by prompting for a command, displaying a period (.):

```
!
.
```

OCP may not be activated in a system which does not have the OCP modules linked-in (i.e., OCP may not be executed as a dynamic task).

OCP is deactivated by issuing the OCP Terminate (TE) command in response to a prompt:

```
.TE.
```

3.3 LUNOs

OCP uses two LUNOs, 1 and 2. LUNO 1 is assigned to the device to which all messages or output is to be directed. LUNO 2 is assigned to the device which contains a file (e.g. object program) being input.

3.4 COMMAND FORMAT AND SYNTAX.

Each command consists of a command word, optionally followed by one or more operands. OCP recognizes a command by the first two letters of the command word; these letters may be followed by additional letters or by blanks. One or more blanks, or a comma, may separate a command word and operands. However, embedded blanks are not allowed within the command word or an operand. More than one command may be entered in a single line, which may contain up to 72 characters and must be terminated by a carriage return (NEW LINE on 911 or 913 Video Display Terminal). When more than one command is entered on one line, each command must be terminated with a period (the period may be omitted following the last command on the line). When an error is detected in a command, any subsequent command on that line is ignored. Two consecutive commas are interpreted as a null operand, which may be used to omit an optional operand. When a null operand is entered for a numeric operand, the null operand takes the value of zero. When a null operand is entered for a character operand, the null operand takes the value of a

blank. If a command generates printout to a hardcopy device such as a 733 ASR, the printout can be aborted by pressing the ESCape key. If a line is entered in error and if the carriage return has not yet been entered, the operator may press the RUB OUT key. This deletes the entry from the OCP input buffer, and the user can then enter a correct command.

NOTE

All numeric OCP inputs are assumed to be hexadecimal numbers.

The following examples illustrate the manner in which OCP interprets equivalent commands entered in different ways. The commands in the examples are equivalent and valid.

.DMEM,0000,0010.	The complete command word is entered with two hexadecimal operands separated by commas and terminated with a period.
.DM 0 10	The command word is entered in two-letter form; blanks are used as separators; leading zeros are omitted; and the period is omitted. (Period is not omitted when another command is entered on the same line.)
.DMEM, 0 10	The complete command word is entered; an additional blank is entered; a comma is used as a separator in one place and a blank in another.
.DM,, 10	The command word is entered in two-letter form, and first operand is entered as a null operand.
.DMXXX,0000 10	The first two letters of the command word are followed by other letters; a blank is used as a separator.
.DM 0 10 XYZ 55	This is the same as the second example except for additional operands, which are ignored.

In subsequent definitions of OCP commands, the following notation is used.

- Angle brackets < > enclose items supplied by the user
- Brackets [ ] enclose optional items
- Braces { } enclose alternative items, one of which must be entered
- An ellipsis (. . .) indicates that the preceding item may be repeated
- Items shown in capital letters must be entered as shown

3.5 OCP COMMANDS

The OCP commands are divided into the following groups: task support commands, debugging commands, I/O utility and status commands, time and data commands, and the termination command. Table 3-1 shows every OCP command and its syntax.

Table 3-1. Syntax of OCP Commands

OCP Task Support Commands

	AL / ALUNO	,<luno>,<pathname>.
	RL / RLUNO	,<luno>,
	LP / LPROG	,<pathname>[,[<priority>] [P]]
	EX / EXECUTE	,<task id>[,<parm1>[,<parm2>]].
Available only with multiple dynamic task support.	IT / ITASK	,<pathname>,<task id> [,<priority>] [,<procedure id>] [,P] [,N]
	IP / IPROC	,<procedure id>.
	DT / DTASK	,<task id>.
	DP / DPROC	,<procedure id>.

OCP Debugging and Error Recovery Commands

DM / DMEN	,<starting address>[,<ending address>].
LM / LMEN	,<address>,<value>[,<value>] ...
SB / SBKPT	,<address>.
CB / CBKPT	[,<address>].
AD / ADD	,<value>,<value>.
SU / SUB	,<value>,<value>.
JM / JMP	,<address>,<address>.

Table 3-1. Syntax of OCP Commands (Continued)

OCP Debugging and Error Recovery Commands (Continued)

{DW / DWKSP}	,<task id>.
{KT / DTASK}	,<task id>.
{KI / KIO}	,<luno>.
{TR / TRACE}	[,<address>].

OCP I/O Utility and Status Request Commands

{RE / REQIND}	,<luno>.
{FS / FSPACE}	,<luno>,<number>.
{BS / BSPACE}	,<luno>,<number>.
{ST / STASK}	[,<task ID>].
{SI / SIO}	[,<luno>]
{SP / SPROC} (Available only with multiple dynamic task support.)	[,<procedure id>].

OCP Time and Date Commands

{ID / IDATE}	,<year>,<month>,<day>,<hour>,<minute>.
{TI / TIME}	.

OCP Termination Command

{TE / TERMINATE}	.

Each group of commands is discussed in detail in a subsequent paragraph.

3.5.1 **OCP TASK SUPPORT COMMANDS.** These commands are processed by either module OCPLRT or module DOCPLRT, and enable the user to assign and release LUNOs, load programs into the dynamic task area, and execute programs, through a terminal. DOCPLRT should be used to support multiple dynamic tasks.

3.5.1.1 **Assign Logical Unit Number (AL or ALUNO).** The Assign Logical Unit (AL or ALUNO) command assigns a logical unit to a pathname. The syntax for the command is as follows:

$$\left\{\begin{matrix}\text{AL} \\ \text{ALUNO}\end{matrix}\right\} \text{,<luno>, <pathname>.}$$

The LUNO operand is a hexadecimal number from 0 through FE_{16}. In systems that include OCP, LUNO 0 is a system LUNO and may not be assigned by the user. When a previously assigned LUNO is reassigned, the previous assignment is released unless a task is performing I/O to the LUNO. When a task is performing I/O to the LUNO, the LUNO is neither released nor reassigned and an error message is printed.

The following are examples of ALUNO commands:

AL,10,CR.	Assign LUNO 10_{16} to the card reader.
ALUNO,11,LP.	Assign LUNO 11_{16} to the line printer.
AL,10,:FILE/SYS.	Assign LUNO 10_{16} to a disc file.

Error messages are discussed in paragraph 3.6 and in Tables 3-2 and 3-3.

Error messages 1 through 6 (Table 3-3) apply to the AL or ALUNO command.

3.5.1.2 **Release Logical Unit Number (RL or RLUNO).** The Release Logical Unit (RL or RLUNO) command releases the assignment of a LUNO to a physical device or file. An attempt to release a LUNO to which a task is performing I/O results in an error message. The syntax for the command is as follows:

$$\left\{\begin{matrix}\text{RL} \\ \text{RLUNO}\end{matrix}\right\} \text{,<luno>.}$$

The LUNO operand specifies a LUNO to be released. The following is an example of an RLUNO command:

RL,1A.	Release LUNO $1A_{16}$.

Error messages 1 through 4 (table 3-3) apply to this command.

3.5.1.3 Load Program (LP or LPROG). The Load Program (LP or LPROG) command reads the object code on the specified device and installs it in the dynamic task area. When a task that was previously installed in the dynamic task area has not terminated, the current task is not installed and an error message is printed. The snytax for the command is as follows:

$$\left\{\begin{matrix}\text{LP}\\\text{LPROG}\end{matrix}\right\}\ \text{,<pathname>[,[<priority>][,P]]}.$$

The pathname operand is the name of the input device or file appropriate for the module that contains the object code for the task. OCP assigns LUNO 2 to the device. The priority operand is the priority level, 0 through 3, of the task. When the priority operand is omitted, priority level 3 is assigned to the task. The third operand is the letter P to specify that the task executes in the privileged mode. When the third operand is omitted, the task executes in the nonprivileged mode.

The task identifier (ID) of a task installed with the LRROG command is 10_{16}.

When loading from a cassette, the LPROG command processor does not perform a rewind operation prior to reading the module. It is necessary for the user to position a cassette to the desired object module. When loading from a diskette file, the file is automatically rewound.

The following examples show use of LPROG commands:

LPROG,CS1.	Read the object module (i.e., the program or task) from the cassette in cassette drive 1 and install the module in the dynamic task area at priority level 3.
LP,CR,2.	Read the object module from the card reader and install the module in the dynamic task area at priority level 2.
LP,DSC2:IBMUTL/SYS,3,P.	Read the object module from the diskette in diskette drive 2 and install the module in the dynamic task area at priority level 3 as privileged.

Error messages 1, 2, 5, 6 and 7 through 11 (table 3-3) apply to this command.

3.5.1.4 Execute Task (EX or EXECUTE). The Execute Task (EX or EXECUTE) command places the specified task in execution. Execution begins when the task becomes the oldest task on the active list for its priority level, and no task at a higher priority level is active. The syntax for the command is as follows:

$$\left\{\begin{matrix}\text{EX}\\\text{EXECUTE}\end{matrix}\right\}\ \text{,<task id>[,<parameter1>[,<parameter2>]]}.$$

The task ID operand is a two-digit hexadecimal operand. The task ID of the task installed in the dynamic task area is 10_{16}. Task IDs of resident tasks are assigned when the system is generated. The parameter operands are optional task parameters, each of which is a hexadecimal number of up to four digits. When the operands are omitted, TX990 supplies zeros. The task obtains the parameters by executing a Get Parameter supervisor call, described in Section IV.

The following are examples of EXECUTE commands:

EX,10.	Execute task 10_{16}, installed in the dynamic task area.
EX,11,4845,4C50.	Execute task 11_{16}, and passes the characters HELP to the task.
.EXECUTE,10,1C.	Execute the task installed in the dynamic task area, and pass the value $1C_{16}$ (28) to the task.

Error messages 1 and 12 (table 3-3) apply to this command.

3.5.1.5 Install Task (IT or ITASK). The Install Task command loads an object module from a sequential file into the dynamic task area. The syntax of the command is as follows:

{IT / ITASK} ,<pathname>,<task id>[,[<priority>][,[<procedure id>][,P][,N]]]

The <pathname> is the name of the sequential file or device which contains the object module of the task being installed, and is required.

The required <task id> parameter is the ID to be assigned to the installed task. Valid IDs are in the range of 10_{16} to EF_{16}.

The optional <priority> parameter is the priority (0 to 3) at which the task will execute. The default value is 3, the lowest priority.

<Procedure id> is the ID of a previously installed procedure which is to be attached to the task when loaded. If no value is given, no procedure is attached to the task.

The letter P is used to specify that the task should execute in privileged mode; if not used, the task is nonprivileged.

The letter N is used to prevent the input file <pathname> from being rewound before the task is loaded. If N is not used, the file is rewound.

The following examples show the use of the Install Task command:

ITASK, DSC: TASK/OBJ, 30, 3, 10, P, N

Read the object module from file :TASK/OBJ and install it as task 30_{16} with priority 3. Attach procedure 10_{16} and make the task privileged. Do not rewind the file.

IT, CS1, CB.

Read the object module from CS1 and install it as task CB_{16}. The task is nonprivileged, has no attached procedures, and has priority 3.

3.5.1.6 Install Procedure (IPROC). The Install Procedure (IPROC) command reads the specified file and installs it in the dynamic task area. The syntax for the command is as follows:

{IP / IPROC} ,<pathname>,<procedure id>

The <pathname> is the name of the sequential file or device which the object module of the procedure being installed, and is required.

The <procedure id> specifies the procedure to be installed with the IPROC command. If a procedure with the same id is already installed, the new procedure is not installed and an error message is returned.

3.5.1.7 **Delete Task (DTASK).** The Delete Task command deletes a task from memory and returns the memory occupied by the task to the available memory pool in the dynamic task area. The syntax for the command is as follows:

{DT / DTASK} <task id>.

The task previously installed with an ITASK command using the same <task id> is deleted. If a task with the specified ID is not installed in the dynamic task area, no task is deleted, and an error message is printed. When a procedure is attached by an ITASK command to the task being deleted, the task is detached from the procedure. A task linked with the system may not be deleted.

3.5.1.8 **Delete Procedure (DPROC).** The Delete Procedure command deletes a procedure from memory and returns the memory to the available memory pool in the dynamic task area. The snytax for the command is as follows:

{DP / DPROC} <procedure id>.

The procedure previously installed with an IPROC command using the <procedure id> is deleted unless one or more tasks are attached. When one or more tasks are attached or a procedure with the specified ID is not installed, the procedure is not deleted, and an error message is printed.

3.5.2 **OCP DEBUGGING AND ERROR RECOVERY COMMANDS.** These OCP commands are processed by Command Processor module OCPSLD and OCPTLD, and enable the operator, by use of the system console keyboard, to perform some real debugging operations; dumping and loading memory, which allows the user to patch his task or to examine different variables in memory; setting and clearing run-time breakpoints in a task; adding and subtracting hexadecimal addresses; calculating JMP instruction displacements; and dumping the current workspace of an executing task. These commands are described in the following paragraphs.

3.5.2.1 **Dump Memory (DM or DMEM).** The Dump Memory (DM or DMEM) command causes OCP to print the contents of specified sequential memory locations on the device to which LUNO 1 is assigned. The syntax for the command is as follows:

{DM / DMEM} ,<starting address>[,<ending address>].

The starting address operand is a one- to four-digit hexadecimal address of the first memory location to be displayed. The optional ending address operand is a one- to four-digit hexadecimal address of the last memory location to be displayed. When the ending address operand is omitted, the ending address is equal to the starting address. Contents of eight memory words are printed on one line, following the address of the first memory location displayed. The contents are displayed in hexadecimal representations and in ASCII character representations, with nonprinting characters printed as periods. OCP displays a multiple of eight words; when the ending address is omitted or when the ending address is greater than the starting address by 16 or less, one line is printed. Similarly, the last line printed includes the contents of the ending address and any additional words required to fill the line. When the starting address is an odd value, the next lower even address is used.

The following is an example of a DMEM command and the resulting display of memory contents:

.DM,15,34. Display contents of memory locations 0015_{16} through 0034_{16}.

```
0014 = 07BA  0796  0568  05E2  0568  05E2  0568  05E2  .. .. .. .. .. .. .. ..
0024 = 0572  05B0  057C  05BC  0568  05E2  0568  05C8  .. .. .0 .. .> .. .. II
0034 = 0568  05E2  0568  05E2  0590  05D4  0568  05E4  .. .. .. .. .. .T .. ..
```

Error message 1 (table 3-3) applies to this command. On a computer equipped with TILINE*, an attempt to address a nonexistent memory address terminates OCP. On other computers, the command is executed, but gives unpredictable results.

3.5.2.2 **Load Memory (LM or LMEM).** The Load Memory (LM or LMEM) command places values in memory at specified locations. The syntax for the command is as follows:

$$\left\{\begin{matrix}\text{LM}\\ \text{LMEM}\end{matrix}\right\},\langle\text{address}\rangle,\langle\text{value}\rangle[,\langle\text{value}\rangle.\ldots]$$

The address operand is a one- to four-digit hexadecimal address into which the first value operand is placed. The value operand is a one- to four-digit hexadecimal representation of the value to be placed in a word of memory. When additional values are entered, they are placed in successive addresses. When the address operand is an odd address, the value is at the next lower even address. The following is an example of an LMEM command:

LM,0100,FFFF,1284. Place the values $FFFF_{16}$ and 1284_{16} in memory words at addresses 0100_{16} and 0102_{16}, respectively.

Error message 1 (table 3-3) applies to this command. On a computer equipped with TILINE*, an attempt to address a nonexistent memory address terminates OCP. On other computers, the command is executed but the results are unpredictable.

3.5.2.3 **Set Breakpoint (SB or SBKPT).** The Set Breakpoint (SB or SBKT) command sets a breakpoint in a task. A breakpoint may be set in any resident task or in a task that has been installed in the dynamic task area, whether or not the task is active. Setting a breakpoint consists of replacing the contents of the specified breakpoint location with a single-instruction loop (JMP $).

CAUTION

Do not mistakenly set the breakpoint outside of the task area, or a memory area other than the task area may be destroyed.

As many as four breakpoints may be active at any one time. The syntax of the command is as follows:

$$\left\{\begin{matrix}\text{SB}\\ \text{SBKPT}\end{matrix}\right\},\langle\text{address}\rangle.$$

The address operand is a one- to four-digit hexadecimal address at which the breakpoint is set. The following is an example of an SBKPT command:

SB,048C. Set a breakpoint at the instruction at address $048C_{16}$.

*TILINE is a registered trademark of Texas Instruments Incorporated.

Error messages 1, 6, and 13 (table 3-3) apply to this command. On a computer equipped with TILINE, an attempt to set a breakpoint in a nonexistent memory address terminates OCP. On other computers, the command is executed, but the results are unpredictable.

The user must be careful to set a breakpoint at the address of a single-word instruction, or at the address of the first word of a multiword instruction. Effects of setting breakpoints at other locations are unpredictable.

3.5.2.4 Clear Breakpoint (CB or CBKPT). The Clear Breakpoint (CB or CBKPT) command clears one or all breakpoints in a task or system. A breakpoint may be cleared in any resident task or in a task that has been installed in the dynamic task area, whether or not the task is active.

When a breakpoint occurs, a CBKPT command must be executed to continue execution of the task. Clearing a breakpoint consists of restoring the original contents of the breakpoint location. However, when clearing a breakpoint is requested for a breakpoint location that no longer contains a single-instruction loop (JMP $) the contents of memory are not altered. The syntax for the command is as follows:

$$\begin{Bmatrix}\text{CB} \\ \text{CBKPT}\end{Bmatrix} \text{,<address>.}$$

The address operand is a one- to four-digit hexadecimal address of a breakpoint to be cleared. When the address is omitted, all current breakpoints are cleared. The following is an example of a CBKPT command:

```
CBKPT,048C.
```

Clear breakpoint at address $048C_{16}$.

Error messages 1, 12, and 14 (table 3-3) apply to this command.

3.5.2.5 Add (AD or ADD). The Add (AD or ADD) command provides the sum of two values. The syntax of the command is as follows:

$$\begin{Bmatrix}\text{AD} \\ \text{ADD}\end{Bmatrix} \text{,<value>,<value>.}$$

The value operands are one- to four-digit hexadecimal numbers, and are added using the A machine instruction. The addition follows the rules that apply to the A instruction, and the result is not modified by the state of the Carry or Overflow bits following the operation. The following are examples of ADD commands and resulting sums:

```
 ADD,300,700.
0A00
```

Adds 300_{16} to 700_{16} and prints the sum $0A00_{16}$.

```
 AD,7FFF,8000.
FFFF
```

Adds $7FFF_{16}$ to 8000_{16} and prints the sum $FFFF_{16}$, the maximum value that is represented without an overflow.

```
 AD,8000,8000.
0000
```

Add 8000_{16} to 8000_{16} and prints the four least significant digits of the sum, 0000.

Error message 1 (table 3-3) applies to this command.

3.5.2.6 Subtract (SU or SUB). The Subtract (SU or SUB) command subtracts the first of two specified values from the second, and prints the difference. The syntax for the command is as follows:

$\left\{\begin{matrix}\text{SU}\\\text{SUB}\end{matrix}\right\}$,<value>,<value>.

The first value operand (subtrahend) is the value to be subtracted from the second value operand (minuend). Both operands are one- to four-digit hexadecimal numbers, and are subtracted using the S machine instruction. The subtraction follows the rules that apply to the S instruction, and the result is not modified by the state of the Carry or Overflow bits following the operation. The following are examples of SUB commands and resulting differences:

```
 SUB,500,A00.
0500
```
Subtract 500_{16} from $A00_{16}$, and prints the difference, 500_{16}.

```
 SU,FFFF,100.
0101
```
Subtract $FFFF_{16}$ (-1) from 100_{16}, and prints the difference, 101_{16}.

```
 SU,1000,300.
F300
```
Subtract 1000_{16} from 300_{16}, and prints the difference, $F300_{16}$ ($-D00_{16}$).

Error message 1 (table 3-3) applies to this command.

3.5.2.7 Jump Instruction (JM or JMP). The Jump Instruction (JM or JMP) command builds a jump instruction having a displacement corresponding to the specified addresses. The JMP command prints an instruction word with a 1 as the first digit and the displacement in the two least significant digits. An asterisk follows the 1 and represents the second hexadecimal digit of a specific jump instruction; i.e., 0 for JMP, 3 for JEQ, etc. The syntax for the command is as follows:

$\left\{\begin{matrix}\text{JM}\\\text{JMP}\end{matrix}\right\}$,<address>,<address>.

The address operands are one- to four-digit hexadecimal addresses; the first is that of the jump instruction being built; the second address is that of the instruction to which control is to be transferred by the jump instruction. The following are examples of JMP commands and the resulting instructions:

```
 JMP,109A,10B0.
1*0A
```
Build a jump instruction to be placed at address $109A_{16}$ to jump to address $10B0_{16}$.

```
 JM,109A,1000.
1*B2
```
Build a jump instruction to be placed at address $109A_{16}$ to jump to address 1000_{16}.

Error messages 1 and 15 (table 3-3) apply to this command.

3.5.2.8 Dump Workspace (DW or DWKSP). The Dump Workspace (DW or DWKSP) command causes OCP to print the contents of the workspace of a specified task. When the debugging OCP described previously is not included, this command will not execute. OCP also prints the task address and the workspace pointer register contents. The syntax for the command is as follows:

$\left\{\begin{matrix}\text{DW}\\\text{DWKSP}\end{matrix}\right\}$,<task id>.

The task ID operand is the task identifier of the task. The task identifier is a one- or two-digit hexadecimal number assigned when the task was loaded. The following is an example of a DWKSP command and the resulting printout:

```
DWKSP,0D.                    Print contents of workspace of task 0D16.

   ADR: 20F0      WP: 20F6
20F6 = 06BE  06AE  06CA  06AC  06EA  070A  06EA  070A  .> .. .. .. .. .. .. ..
2106 = 05E4  0568  05E4  0568  05E4  0568  05E4  0568  .. .. .. .. .. .. .. ..
```

On the first line, OCP prints the starting address at which the task was loaded, and the contents of the workspace pointer register. On the next two lines, the contents of the 16 words of the workspace are printed in the format of the DM command. The address of the first word on each line is printed as a four-digit hexadecimal number, followed by an equal sign. The address on the first of these two lines is the address that is in the workspace pointer register. To the right of the equal sign are eight four-digit hexadecimal numbers that represent the contents of workspace registers 0 through 7 in hexadecimal representation. The ASCII representation of the same values is printed further to the right, with periods substituted for nonprinting ASCII characters. The contents of workspace register 8 through F are printed in the same format on the next line.

Error messages 1 and 12 (table 3-3) apply to this command.

3.5.2.9 Kill Task (KT or KTASK). The Kill Task (KT or KTASK) command forces termination of an executing task. If end action is specified by the task, it is taken. Otherwise, any I/O operations in progress are terminated in error; all assignments of file-oriented devices to the task are released; and any files that were opened are closed. The syntax for the command is as follows:

```
{KT    } ,<task id>.
{KTASK }
```

The task ID operand is the task identifier assigned when the task was installed, and consists of two hexadecimal digits. The following is an example of a KTASK command:

```
KTASK,21.        Force termination of task 2116.
```

Error messages 1, 12, and 16 (table 3-3) apply to this command.

3.5.2.10 Kill I/O Operation (KI or KIO). The Kill I/O Operation (KI or KIO) command terminates any I/O operation to the specified device. If an I/O operation to the device is in progress, that operation is terminated in error. The syntax for the command is as follows:

```
{KI  } ,<luno>.
{KIO }
```

The LUNO operand is a hexadecimal number, the LUNO assigned to the device on which I/O is to be terminated. When the specified device is file-oriented, the command also closes the file, which releases the assignment of the device to the task. The following example shows a KIO command:

```
KIO,30.          Terminate I/O to the device or file to which LUNO 3016 is assigned.
```

Error messages 1 and 17 (table 3-3) apply to this command.

3.5.2.11 **Trace (TR or TRACE).** Trace (TR or TRACE) displays the current contents of a memory location on the programmer panel lights. When the contents of memory change, the programmer panel lights will also change. It has up to one parameter. This parameter is the hexadecimal memory address. If no parameter is given, the standard display, the Program Counter (PC), will be displayed on the programmer panel. By entering the command with no parameter, the trace is, in effect, turned off.

TRACE,[<memory address>].

3.5.3 **OCP I/O UTILITY AND STATUS REQUEST COMMANDS.** These OCP commands are processed by Command Process Module OCPIOU or DOCPIOU, and enable the operator, by use of the system console keyboard, to perform various I/O utility and status request commands. DOCPIOU should be included to support multiple dynamic tasks.

3.5.3.1 **Rewind Device (RE or REWIND).** The Rewind Device (RE or REWIND) command initiates a rewind operation on a rewindable device or file. The syntax for the command is as follows:

{RE / REWIND} ,<luno>.

The LUNO operation is a hexadecimal number, the LUNO assigned to the device or file to be rewound. When the LUNO is assigned to a device that is not rewindable, the command is ignored. The following is an example of a REWIND command:

RE,2C. Rewind the device to which LUNO $2C_{16}$ is assigned.

Error messages 1, 17, and 18 (table 3-3) apply to this command.

3.5.3.2 **Forward Space (FS or FSPACE).** The Forward Space (FS or FSPACE) command forward-spaces a sequential file or device a specified number of records. When the file or device forward spaces to an end-of-file (EOF) record, it remains positioned at the record following the end-of-file record. The syntax for the command is as follows:

{FS / FSPACE} ,<luno>,<number>.

The LUNO operand is the LUNO to which the device or file to be forward-spaced is assigned. When the LUNO is assigned to a device other than the cassette, the command is ignored. The number, a one- or two-digit hexadecimal value, is the number of records to be forward spaced. The following is an example of an FSPACE command:

FS,1C,5. Forward space the device to which LUNO $1C_{16}$ is assigned 5 records.

Error messages 1, 17, 18, and 19(table 3-3) apply to this command.

3.5.3.3 **Backspace (BS or BSPACE).** The Backspace (BS or BSPACE) command backspaces a sequential file or device a specified number of records. When the cassette tape is at the beginning-of-tape marker, the command is ignored. The syntax for the command is as follows:

{BS / BSPACE} ,<luno>,<number>.

The LUNO operand is the LUNO to which the device to be backspaced is assigned. The number, a one- or two-digit hexadecimal value, is the number of records to be backspaced. The following is an example of a BSPACE command:

BSPACE,2E,10. Backspaces the device to which LUNO $2E_{16}$ is assigned 10_{16} records.

Error messages 1, 17, 18, and 19 (table 5-4) apply to this command.

3.5.3.4 Task Status (ST or STASK). The Task Status (ST or STASK) command causes OCP to print the current status of all tasks. The syntax for the command is as follows:

{ST / STASK} [<task id>].

No operand is required; however, if <task id> is supplied, only the status of that one task will be printed. The following is an example of an STASK command and the resulting printout:

```
. ST .
ID      PRIOR     ADDR      WP      PC      ST      STATE     PROC
OF        01      18F8      18FE    1D38    240F      07
OD        01      37CE                              04
10        01      38EE                              04
A1        02      BCAO                              04        27
        0:02:53       JAN  1,        0
```

OCP prints a heading, followed by a line for each task. In the first column, OCP prints the task ID. OCP prints the priority level of the task in the second column. The next four columns contain four-digit hexadecimal addresses. The first of these, under the heading ADDR, is the address of the task. Next, under the heading WP, is the value to be placed in the workspace pointer register when the task is executed. In the column headed PC is the program counter contents, and in the column headed ST is the status register contents. The contents of the WP, PC, and ST registers are the contents at the most recent execution of the task. The seventh column, headed STATE, contains the task state code used by task management. These codes are listed in Appendix G. The last column, headed *PROC*, is only specified if multiple dynamic tasks are allowed. It contains the procedure identifier if a procedure is attached to the task. It is blank when no procedure is attached to the task.

Task identifiers and priority levels are assigned at system generation, except for tasks in the dynamic task area, which have identifiers and priority levels assigned by the ITASK command.

Task addresses are those into which the tasks were loaded when the system was loaded, or during execution of the ITASK command.

3.5.3.5 Status of I/O (SI or SIO). The Status of I/O (SI or SIO) command-request causes OCP to print the status of all assigned LUNOs or of a particular LUNO. The syntax for the command is as follows:

{SI / SIO} [,<luno>]

No operand is required. The following is an example of an SIO command and the resultant printout:

```
.SI.
LUNO PATHNAME          TASK ID      SYSLUN
F0 DSC                 –            Y
F1 DSC2                –            Y
00 LOG                 –            Y
01 LOG                 –            N
06 DSC:TXCCAT/SYS      –            N
10:11:10 JAN 5, 1977
```

OCP prints a heading, followed by a line for each LUNO and, in the first column, OCP prints the hexadecimal digits of the LUNO. The PATHNAME column contains the device, or file name to which the LUNO is assigned; and the SYSLUN column contains YES for LUNOs designated for system use and NO for other LUNOs. The Task ID column contains hyphens for LUNOs that are not assigned to tasks, or the task identifier of tasks to which LUNOs are assigned. A LUNO for a file-oriented device is assigned to a task from the time the task issues an Open supervisor call. This call specifies the LUNO until the task terminates or issues a Close supervisor call for the LUNO. A LUNO for a record-oriented device is assigned to a task only during an I/O operation. If the status is requested for a LUNO which has not been assigned, the header line only will be printed.

```
.SI,20.

LUNO PATHNAME                TASK ID      SYSLUN
```

3.5.3.6 **Procedure Status (SP or SPROC).** The Procedure Status (SPROC) command causes OCP to print the current status of procedures. The syntax for the command is as follows:

```
SP
SPROC      [<procedure id>].
```

The status of the specified procedure will be printed. When the <procedure id> is omitted, the status of all procedures will be printed. The following is an example of a SPROC command and the resulting printout.

```
. SP .

ID      ADR      # TASKS

27      BE60       01

        0:02:37    JAN   1,     0
```

OCP prints a heading, followed by a line for each procedure. In the first column, OCP prints the procedure identifier. Column two contains the address at which the procedure was loaded. OCP prints the number of tasks to which the procedure is attached in the third column. Procedure identifiers are assigned by the IPROC command. The number of tasks to which the procedure is attached is incremented each time an ITASK command specifies attachment of that procedure and is decremented each time a DTASK command deletes a task to which that procedure had been attached.

3.5.4 **OCP TIME AND DATE COMMANDS.** The OCP date and time commands are:

- Initialize Date and Time (ID or IDATE)
- Print Time and Date (TI or TIME)

These OCP commands are processed by Command Processor Module OCPTAD and enable the operator, by use of the system console keyboard, to initialize time and date for the Operating System. The above listed commands are described in the following paragraphs.

3.5.4.1 **Initialize Date and Time (ID or IDATE).** The Initialize Date and Time (ID or IDATE) command initializes the date and time values for the system. The syntax for the command is as follows:

{ID / IDATE} ,<year>,<month>,<day>,<hour>,<minute>.

The year operand is the four-digit decimal number of the years 1975 through 1999, and the month operand is the decimal number of the month, 1 through 12. The day operand is a one- or two-digit decimal number, 1 through 31, and the hour operand is a one- or two-digit decimal number, 0 through 23. The PM hours are specified by the sum of 12 and the hour. The minute is the decimal number of the minute, 0 through 59. The second is set to zero when the command is entered. After the command is entered, the time and date will be written to the log for verification. The following example shows an IDATE command:

```
.ID,1976,2,12,17,29.          Initialize the time and date to 5:29 PM, February 12, 1976.

5:29:00  FEB 12, 1976
```

Error message 1 (table 3-3) applies to this command.

3.5.4.2 **Print Time and Date (TI or TIME).** The Print Time and Date (TI or TIME) command causes OCP to print the time and date. The syntax for the command is as follows:

{TI / TIME} .

The command requires no operands. The following example shows a TIME command and the resulting output:

```
TI.
9:29:12  FEB 13, 1976
```

When the date and time have not been initialized, the time printed is the elapsed time from the most recent loading of the Operating System.

3.5.5 **OCP TERMINATION (TE) COMMAND.** The syntax for the TErminate (TE) command is as follows:

$$\left\{\begin{matrix}\text{TE} \\ \text{TERMINATE}\end{matrix}\right\}$$

The command requires no operands.

3.6 ERROR MESSAGES

OCP prints two categories of error messages. The messages in the general category are shown in table 3-2. These messages are not related to any specific command. The messages in the operand category apply to one or more operands of specific commands. These messages are listed in table 3-3 and are numbered to allow them to be associated with the commands to which they apply. The numbers do not represent an error code, but are arbitrarily assigned for identification purposes.

When more than one OCP command is entered on a line, and an error occurs, the command in error (or that caused the error) and all subsequent commands in the statement must be entered again. It is sometimes necessary to supplement the error message information with information obtained by executing other OCP commands (ST, SI, DM, DW, etc.) to determine which commands have executed correctly. System generation documentation provides information about the tasks actually in the system, their identifiers, and their priority levels. It is also helpful to enter the commands individually so that any error messages are printed following entry of the command. However, the user must exercise care to avoid reexecuting a command that executed correctly and will provide undesirable results if it is executed again.

When LUNO 1 is assigned to a device that is off line, or when LUNO 1 is not assigned and a command is used which generates a listing (DMEM,STASK,SIO), OCP ignores the command and prompts the user with a period(.).

Table 3-2. OCP General Error Messages

Message	Meaning	Recovery
* INVALID COMMAND *	The command word is not valid.	Check the command word, and reenter correctly.
I/O ERROR, TERMINATED	An I/O error was detected during reading of the command.	Input from the system console was not received correctly or the device timed out.
OUTPUT ERROR	An error was detected during output and execution of the command has been terminated.	This message is printed when LUNO 1 has not been assigned. Check device for errors.
UNDEFINED ERROR	TX990 returned an error code to OCP that is not recognized by OCP.	This is a system error. Make another attempt to enter the command. If error reoccurs, reinitialize the system.
MISSING SYS MSG	OCP could not locate the text for an OCP error message.	OCP is incorrectly configured or incorrectly programmed. First verify that the proper OCP modules were linked with the system, and relink and reload the system if they were not. If all required modules were linked, and these modules were supplied by Texas Instruments, contact your Texas Instruments representative. If user-supplied OCP modules are involved, refer to error message information in paragraph 6.2.2 and verify the coding of the OCP module.

Table 3-3. OCP Operand Error Messages

Message Number	Message	Meaning	Recovery	Applies to
1	OPERAND ERROR (S)	One or more operands are invalid.	When a required operand has been omitted, enter the complete command. When a numeric operand is required, but a nonnumeric operand was entered, enter the command with the correct type of operand.	AL, RL, LP, EX, DM, LM, SB, CB, AD, SU, JM, KT, KI, RE, FS, BS, ID, DW.
2	LUNO IN USE	The LUNO is busy and cannot be released or reassigned until the LUNO becomes available.	If a record mode I/O operation to the LUNO is in progress, enter the command again after the opertion has completed. If the LUNO is assigned to a file-oriented device, and has been opened, enter the command again after the LUNO has been closed.	AL, RL, LP.
3	BAD LUNO	Invalid LUNO specification.	Enter the command with a valid LUNO, 0 through FE_{16}.	AL, RL.
4	NO! SYSTEM LUNO	A system LUNO cannot be reassigned or released.	Enter the command with the correct LUNO. When the system inludes OCP LUNO 0 is a system LUNO.	AL, RL.
5	BAD DEVICE	Invalid device name.	Enter the command with one of the device names assigned when the system was generated.	AL, LP.
6	TABLE AREA FULL	The table area used by the command is full.	When the message entry of an AL or LP command, release an unused LUNO, and enter the command again. Alternatively, generate a new TX990 that contains more LUNO blocks. When the message follows entry of an SB command, clear a breakpoint and enter the command again.	AL, LP, SB.

Table 3-3. OCP Operand Error Messages (Continued)

Message Number	Message	Meaning	Recovery	Applies to
7	CHECKSUM ERROR	Checksum error occurred while reading load module.	The device that reads the module is unable to read a record accurately. The last record read is the record in error. Make another attempt to read the record, or obtain another module and load. This error is normal when a record of the object module has been modified without correcting the checksum of the record. To override the error, enter the LP command again without repositioning the medium on which the object module resides.	LP.
8	I/O ERROR	Error occurred while reading load module.	Position the medium at the first record and enter the command again to read the module again, or obtain another module and load.	LP.
9	WON'T FIT	The program being loaded is too large for the dynamic task area.	The size of the dynamic task area is printed by STASK. To execute a task larger than that size, increase the size of the dynamic task area by relinking the system with fewer modules or by increasing the size of memory.	LP.
10	CAN'T BID LOADER	The loader module is currently in use or is not included in the system.	When current loading has been completed, enter the command again. If the loader is not in the system, relink the system to include the loader.	LP.
11	TASK RUNNING	A user task in the dynamic task area is currently executing.	When the task completes execution, enter the command again. Alternatively, enter a KTASK command to force termination of the task, and enter the command again.	LP.

Table 3-3. OCP Operand Error Messages (Continued)

Message Number	Message	Meaning	Recovery	Applies to
12	NOT FOUND	The specified task cannot be found.	When the task cannot be found because the user has entered the identifier incorrectly, enter the command with the correct identifier. When the identifier is correct, but the task was not identified to the system or linked to the system, generate and/or link the system correctly.	EX, CB, KT, DW.
13	ALREADY SET	A breakpoint has already been set at this location, or the location contains a JMP $ instruction.	When the breakpoint was previously set, the command does not need to be entered again. When the location contains JMP $ in error, enter the correct instruction with an LMEM command. When the JMP $ instruction is the correct contents, no further command is required. However, the task will not execute beyond that point.	SB.
14	NOT JMP $	The location had a breakpoint set but does not contain a JMP $ instruction. Contents of the location are not altered, but breakpoint is removed from the table.	The JMP $ instruction has been overlayed as a result of a programming error or by loading a new task without clearing active breakpoints. Enter a LMEM command to correct the contents if not correct.	CB.
15	OUT OF RANGE	The address in the second operand cannot be reached with a JMP instruction.	Use another instruction or combination of instructions instead of a JMP instruction. Alternatively, reorder the segments of code so that the destination of the instruction is within the range of a JMP instruction.	JM.
16	NOT RUNNING	The specified task is not running.	No recovery necessary.	KT.
17	BAD LUNO	The specified LUNO is invalid or not assigned.	Enter the command again with the correct LUNO. If in doubt about LUNO assignment, enter an SIO command.	RE, FS, BS.

Table 3-3. OCP Operand Error Messages (Continued)

Message Number	Message	Meaning	Recovery	Applies to
18	RESOURCE NOT AVAILABLE	The specified LUNO is assigned to a task.	When the task has completed, enter the command again. Alternatively, enter a KTASK command to force termination of the task and enter the command again.	RE, FS, BS.
19	END-OF-FILE	An EOF was detected during the requested operation.	The operation ceases when the EOF is detected. No recovery necessary.	FS, BS.
20	ID IN USE	The task or procedure ID specified is already used.	Select another ID.	IT, IP.
21	NO! TASKS ATTACHED	The procedure being deleted has tasks attached.	Delete the attached tasks.	DP.
22	BAD ID	An invalid task ID or attached procedure ID was selected.	Select another ID.	IT.

SECTION IV

CONTROL PROGRAM

4.1 INTRODUCTION

The TX990 Control Program provides a simple interface which allows a terminal operator/user to load and execute programs. It can also be used to pass parameters to a program being executed. The following paragraphs describe how to use the Control Program, as well as the mechanism used to pass parameters to initiated tasks (programs).

4.2 ACTIVATING AND DEACTIVATING THE CONTROL PROGRAM

The Control Program, like OCP, may not be executed unless it is linked in with the TX990 operating system. Activation of the Control Program is different between systems which include OCP and systems which do not.

In an operating system which does not include OCP, the control program is activated by entering an exclamation point (!) at an terminal.

In a system which does include OCP, the Control Program must be activated through OCP:

1. Enter ! at a terminal, activating OCP.
2. Respond to the OCP command prompt (i) as shown:

   . EX, 16. TE.

3. The above command executes task 16_{16} (the Control Program) and terminates OCP.

After the Control Program is executed, the following printout or display is presented at the system console:

```
TXDS   936215   **   152/77   1:05

PROGRAM:
```

The previous display tells the operator that the Control Program is in execution and that the operator may respond to the PROGRAM: prompt by specifying the program to be loaded and/or executed. The display heading indicates the name of the monitor (TXDS), the part number of the software, the revision status (** = no revision, *A = 1st revision, *B = 2nd revision, etc.), and the date and time of day that the program was loaded (152/77 = 152nd day of 1977).

The Control Program may be terminated by entering only a carriage return in response to the PROGRAM: prompt.

4.3 LUNOs

The Control Program assigns LUNO 2 to a pathname given in response to the PROGRAM: prompt (i.e., the file or device from which the desired program is to be loaded).

4.4 OPERATOR INTERACTION

The Control Program assists in program loading and execution by printing out or displaying prompts on the system console, sequentially, as follows:

```
PROGRAM:
  INPUT:
 OUTPUT:
OPTIONS:
```

The following paragraphs describe the prompts and user responses.

4.4.1 PROMPT RESPONSES. The Control Program prompts the user to enter the program pathname, input pathname, output pathname, and option-selections. ID or It checks each pathname for syntax. If the syntax is not correct, it will prompt the user again. After all of the responses to the prompts are entered, the Control Program loads and/or executes the specified program.

4.4.1.1 PROGRAM: Prompt. The operator's response to the PROGRAM: prompt is used for specifying either the pathname of the file containing the program to be loaded and executed, or the ID of a program in memory.

Only one pathname can be entered in response to the PROGRAM: prompt. When the program is to be loaded as a privileged task (enabling the task to execute certain supervisor calls), the user must enter the pathname followed by a ",P". A task, when not linked with the operating system, can be made privileged when it is loaded or by issuing a Make Task Privileged SVC at execution. All tasks linked with TX990 are privileged. When the task is loaded, it is assigned task ID 10_{16}.

Should the user desire to execute a task already in memory, the task ID, preceded by a hexadecimal sign (>) must be entered in response to the PROGRAM: prompt. For example, after the TXEDIT utility program has been loaded into memory, it can be reexecuted as follows:

```
TXDS  936215  **  010/77  2:05

PROGRAM:  >10
  INPUT:  DSC:TASK2/SRC
 OUTPUT:  DSC:SCRATCH/SRC
OPTIONS:  (carriage return)
```

4.4.1.2 INPUT: Prompt. The operator's response to the INPUT: prompt is used to specify the pathname(s) of input information needed by the program during its execution. The operator can enter zero to three input pathnames separated by commas. The Control Program will check each parameter for syntax. If the syntax is wrong, the Control Program will prompt the user again. The user must enter the entire line again.

4.4.1.3 OUTPUT: Prompt. The operator's response to the OUTPUT: prompt is used to specify the pathname for storage of the output information resulting from execution of the program. Up to three pathnames (separated by commas) can be entered in response to the OUTPUT: prompt.

4.4.1.4 OPTIONS: Prompt. The operator's response to the OPTIONS: prompt is used to specify the option(s) selected from the total alternative options available for the program which is to be loaded and executed.

4.4.2 DEFAULT VALUES. Except for the PROGRAM: prompt, default values for the Control Program prompts are determined by the program being executed.

If the user intends to execute a task already in memory without reloading it, he must enter the task ID; no default value is allowed.

If the user intends to load the program from a device (not a file), the device name must be specified (i.e., there is no default value). If a file pathname is entered (see Section I on pathnames), the user may leave both the device/volume name and the file extension fields unspecified.

When a PROGRAM: pathname does not specify the diskette volume name or drive, the Control Program starts a device-file search beginning with the diskette drive that is the default-substitute defined during system generation. For a standard TI-supplied system, the default-substitute is DSC. If the file is not on the diskette of the first default diskette transport drive, the Control Program will concatenate a 2 to DSC and the file search would then proceed to DSC2. In the same manner, the search continues to DSC3 and to DSC4. The search is only effective when the diskette default-substitute is the main diskette drive and when its device-name identifier is comprised of three characters, (i.e., DSC or any other three characters). It should also be noted that whenever the user specifies the device-name identifier in response to the PROGRAM: prompt, only the specified device (e.g., the specified diskette transport drive) is searched.

When the user enters only a slash (/) as the extension field in the PROGRAM: prompt pathname, the extension will default to SYS and SYS will be substituted into the pathname before any drives are searched. If neither the extension nor a slash is entered, the extension is assumed to be blanks.

4.4.3 SPECIAL KEYBOARD CONTROL KEYS. The special keyboard control keys are described as follows:

1.	RUB OUT/DELETE LINE	Allows the operator to reenter a parameter. Pressing the RUB OUT key causes a line feed followed by a carriage return. The operator may then enter the line again.
2.	CONTROL H/Back Arrow	Allows the operator to backspace by character and correct a typing error.
3.	Carriage Return/NEW LINE	Causes TXDS Control Program to terminate if the carriage return or NEW LINE was the only entry in response to the PROGRAM: prompt, otherwise terminates a prompt line entry.
4.	ESCAPE/RESET	If an ESCAPE or RESET is entered during a print out, the TXDS Control Program terminates.
5.	,	Causes a default to be activated when entered as the response to the INPUT: or OUTPUT: prompts.
6.	&	In any prompt line, pressing the & key as the first character in the response causes the TXDS Control Program to restart with the PROGRAM: prompt.

7. * When entered after a prompt line entry, in place of a carriage return, permits the next prompt line to be entered without being prompted by the TXDS Control Program. When a prompt line is terminated with an asterisk (*) followed by a carriage return, no more prompts are given and default-substitutes are made by the utility program for those pathnames not entered. The experienced user can enter all or several of the parameters on one prompt line.

The following examples utilize the asterisk (*) feature in lieu of the INPUT:, OUTPUT:, and OPTIONS: prompts:

Example 1:

To load the TXEDIT utility program after the TXDS Control Program has been loaded, the asterisk (*) is used as presented in the following example:

```
TXDS   936215   **   010/77   2:05

PROGRAM:   DSC:TXEDIT/SYS*DSC:TASK2/SRC*DSC:SCRATCH/SRC*
```

(where DSC:TASK2/SRC is the INPUT: pathname; DSC:SCRATCH/SRC is the OUTPUT: pathname; and the OPTIONS: entry is provided by the default-substitution specified in the TXEDIT utility program.)

The above can also be entered as follows:

```
TXDS   936215   **   010/77   2:05

  PROGRAM:      DSC:TXEDIT/SYS
    INPUT:      DSC:TASK2/SRC*DSC:SCRATCH/SRC*
```

Example 2:

To load the SYSUTL utility program after the TXDS Control Program has been loaded, the asterisk (*) is used as follows:

```
TXDS   936215   **   101/77   , 2:05

PROGRAM:   :SYSTUL/SYS***CF,:TEMP/OBJ
```

(where the INPUT: and OUTPUT: parameters are null and the OPTIONS: parameter is CF,:TEMP/OBJ.)

NOTE

1. In the above examples, it is necessary to press the carriage return key at the end of the parameter line to cause the program to be loaded and executed.

2. If a parameter line ends with an asterisk (*) and a pathname is not entered for each prompt, then default substitutes are made by the utility program for those pathnames not entered.

Example 3:

The following example utilizes the comma (,) to cause a default-substitution to be made in the OUTPUT: pathname below.

```
TXDS   9326215   **   010/77   2:05

  PROGRAM:    :TXMIRA/SYS
    INPUT:    :TASK1
   OUTPUT:    ,CRT
  OPTIONS:    M800,X,L
```

(where the OUTPUT: pathname defaults to a substitute specified in the TXMIRA Assembler utility program.)

The following example utilizes both the asterisk (*) and the comma (,) special keyboard controls:

To load the TXMIRA Assembler utility program after the TXDS Control Program has been loaded, the asterisk (*) is used as follows:

```
TXDS:   936215   **   010/77   2:05

PROGRAM:   :TXMIRA/SYS*:TASK1*,CRT*M800,X,L
```

(where TASK1 is the INPUT: pathname and where the OUTPUT: pathname is the default-substitute provided in the TXMIRA Assembler utility program.)

4.5 ACCESSING PARAMETERS THROUGH THE CONTROL PROGRAM

The responses to the prompts of the Control Program are entered into a block of memory known as COMMON memory. The task which is being loaded and executed then accesses COMMON memory for the information contained in these responses. The response-information is stored in the bytes of COMMON memory in an organized nammer using the format presented in table 4-1. The programmer, when coding a user utility program or a user applications program, will find it necessary to become familiar with the format of COMMON memory. Access to the response-information in COMMON memory is provided to the programmer by the use of Get COMMON Data supervisor call 10_{16} (see Section 6). Get COMMON Data returns the memory address and the byte-size of COMMON to the task. The Control Program can only execute in an operating system which was generated with at least 170 bytes of COMMON memory. The user must take this into consideration when performing system generation.

A typical example of an operator's response-entries to the TXDS Control Program's prompts is presented below, immediately followed by the hexadecimal and ASCII representation in binary code of the operator's response that is placed in the COMMON memory block.

```
TXDS   936215   *A   010/77   2:05

  PROGRAM:   :TXLINK/
    INPUT:   :TXTST,/,CS1
   OUTPUT:   :TXTST2/OBJ,LP
  OPTIONS:   ITXT,M4000
```

Table 4-1. Byte-Allocation of COMMON Memory

Parameter	Byte(s)	Explanation
PROGRAM:	0-15	To be coded with the same pathname information that is entered in response to a PROGRAM: prompt.

NOTE

The response-entries to the PROGRAM:, INPUT:, and OUTPUT: prompts are placed in byte-groups of 16 bytes each. The device name is entered in the first four bytes, left-justified, and space-filled with zeros. A colon is placed in the fifth byte if the program name is a diskette file name; otherwise a binary zero is placed in the fifth byte. The file name is entered in the sixth through twelfth bytes, left-justified, and space-filled with binary zeros. A slash is placed in the thirteenth byte when a diskette file is to be referenced by the pathname being entered; otherwise a binary zero is placed in the thirteenth byte. The extension is placed in the fourteenth through sixteenth bytes, left-justified, and space-filled with binary zeros. Whenever the device, file, or extension is to be defaulted by the utility or the user's task, the binary field relating to the device, file, or extension will be space-filled with binary zeros. When the total parameter (which includes the device, file, and extension fields) is defaulted, a colon (:) is placed in the fifth byte and a slash is placed in the thirteenth byte and all the fields become space-filled with binary zeros.

INPUT: #1	16-31	To be coded with the same pathname information that is entered for the first INPUT: parameter.
INPUT: #2	32-47	To be coded with the same pathname information that is entered for the second INPUT: parameter.
INPUT: #3	48-63	To be coded with the same pathname information that is entered for the third INPUT: parameter.
OUTPUT: #1	64-79	To be coded with the same pathname information that is entered for the first OUTPUT: parameter.
OUTPUT: #2	80-95	To be coded with the same pathname information that is entered for the second OUTPUT: parameter.
OUTPUT: #3	96-111	To be coded with the same pathname information that is entered for the third OUTPUT: parameter.
OPTION:	112-143	To be coded with the character-entries that the operator entered in response to the OPTIONS: prompt. The characters entered in response to the OPTIONS: prompt will be copies into 112-143. Up to 30 characters can be entered and copied into COMMON memory and following the last character entered is a binary zero.

Table 4-1. Byte-Allocation of COMMON Memory (Continued)

Chaining Pathname	144-159	Used for the chaining pathname, which is the pathname of the next program to be loaded and executed if the chaining flag in byte 160 is set to a nonzero number. The chaining pathname is initialized so that the first four bytes each have a binary zero, the fifth byte has a colon, the sixth through twelfth bytes each contain a binary zero, the thirteenth byte contains a slash, and the fourteenth through sixteenth bytes each contain a binary zero.
Chaining Flag	160	This is the chaining flag byte which is set to a nonzero number by a user program or a utility program when it is desired to chain from the end of one program to the pathname specified in bytes 144-159. The object program which is at the pathname specified in bytes 144-159 is then loaded and executed. One program (a user's task or TXDS utility program) can chain to another by setting the chaining flag in memory (byte 160), placing the access name (i.e., the chaining pathname) for the new program in bytes 144-159 and executing an End-of-Program 16_{16} supervisor call. The INPUT: OUTPUT:, and OPTIONS: prompts can be used as required to pass parameters to the new program.
Batch Mode Flag	161	Set if batch job control stream is in progress. The TXDS Control Program loads and executes the object program which is in the pathname specified in bytes 144-159.
Batch Error	162	Set when a program terminates in error during a batch stream.
Chaining Error	163	Chaining Error Flag. Set when the program chained to terminate is in error.
Default Print	164-167	Default system console print device declared at time of system generation.
Reserved	168-170	Reserved for later enhancements.

The above responses to the prompts are placed into the COMMON memory block as follows:

Byte Address	Hexadecimal Representation (Upper Row) and ASCII Representation (Lower Row)							
0–15	0000 . .	0000 . .	3A54 : T	584C X L	494E I N	4B00 K .	2F00 / .	0000 . .
16–31	0000 . .	0000 . .	3A54 : T	5854 X T	5354 S T	0000 . .	2F20 / b =	0000 . .
32–47	0000 . .	0000 . .	3A00 : .	0000 . .	0000 . .	0000 . .	2F00 / .	0000 . .
48–63	4353 C S	3100 1 .	0000 . .	0000 . .	0000 . .	0000 . .	0000 . .	0000 . .
64–79	0000 . .	0000 . .	3A54 : T	5854 X T	5354 S T	3200 2 .	2F4F / 0	424A B J
80–95	4C50 L P	0000 . .	0000 . .	0000 . .	0000 . .	0000 . .	0000 . .	0000 . .
96–111	0000 . .	0000 . .	3A00 : .	0000 . .	0000 . .	0000 . .	2F00 / .	0000 . .
112–143	4954 I T	5854 X T	2C40 , M	3430 4 0	3030 0 0		UNDEFINED	
144–159	0000 . .	0000 . .	2A00 : .	0000 . .	0000 . .	0000 . .	2F00 / .	0000 . .
160	00 .							
161	00 .							
162	00 .							
163	00 .							
164–167	4C4F L 0	4720 G	This assumes that LOG was defined as system default print device during system generation.					
168–n			Not used.					

4.6 ERROR MESSAGES

Refer to table 4-2 for a list of error messages, the reason for each error, and the recovery method.

Table 4-2. TXDS Control Program Error Messages

Error	Reason	Recovery
nn-BAD PGM LOAD	nn represents error code listed in error appendix D.	Reenter parameter
–BAD PGM LOAD	Can't find object file.	Reenter parameter
nn–CAN'T BID TASK	nn represents the task state code of task 10_{16} listed in state code appendix C.	Reenter parameter
CAN'T GET COMMON-ABORTED	System was configures without COMMON.	Configure a system with 168 bytes of common

SECTION V

PROGRAMMING TASKS

5.1 INTRODUCTION

A program that executes under the TX990 operating system is also referred to as a task. The task consists of a data division and a procedure division. These divisions may be assembled as a single module or as two or more separate modules linked to form a single object module. The structure of a task is described in this section.

This section also includes a description of the programming considerations for writing tasks to be executed under TX990, and detailed descriptions of the supervisor calls by which a user task requests the support of TX990. Also included is a description of task scheculing.

5.2 TASK STRUCTURE AND PROGRAMMING

Logically, a task consists of a data division and a procedure division. Figure 5-1 shows the task structure supported by TX990, and the relationship of the first three words of the task to the data and procedure divisions. The data division contains the workspaces required by the task and all other data structures (tables, supervisor call blocks, buffers, constants, etc.). The procedure division contains the executable code for the task. When the task is assembled as a single module, the logical division of the task into a data division and a procedure division is optional; however, the task should be organized so that the procedure division may be shared with other tasks should it become desirable.

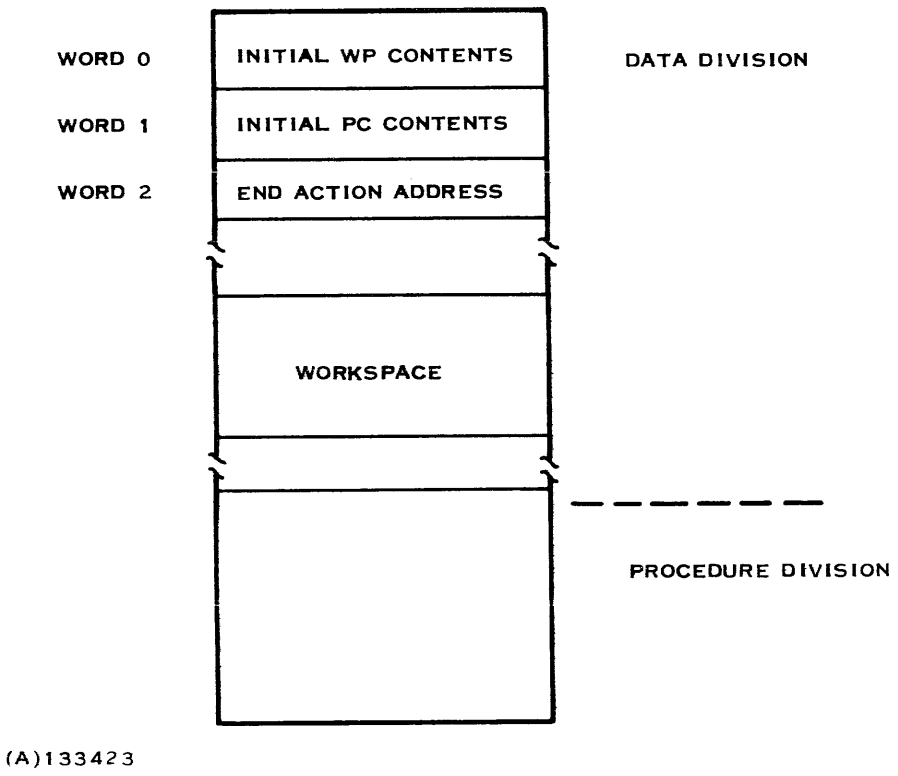


Figure 5-1. TX990 Task Structure

When a procedure division is shared with several data divisions, each combination of a data division with the procedure division is a separate task. The procedure division must be reentrant in this case. General requirements of reentrant programming are described in the *Model 990 Computer TMS9900 Microprocessor Assembly Language Programmer's Guide*, part number 943441-9701. In addition to those requirements, TX990 requires that addressing of data in the data division by the instruction in the procedure be indexed addressing. The user must place the data-division address in one of the workspace registers when the data division is assembled and use this register to index addresses in the procedure.

The user may resolve the addressing problems of a procedure being shared by more than one task in either of two ways. The user may include the dummy origin directive when assembling the procedure, making the data-division relative addresses available to the procedure. Alternatively, the user may link the data divisions and the procedure division. This will result in multiple definitions. The definitions must be identical, if the procedure is to be shared. Therefore, whichever of the multiple definitions is used by the link editor, the result will be correct and the multiple definition messages may be ignored. Any of the tasks that share a procedure may be executed in the dynamic task area. This can be done by linking the data division for that task to the common procedure division, and loading the task into the dynamic-task area.

The first three words of the data division contain the addresses of the initial workspace, the procedure entry point, and the end action entry point, respectively. The contents of the first word are placed in the WP register when the task begins execution. The user places the address of the initial workspace (16-word memory area accessible as 16 workspace registers) in that word. The second word contains the address of the entry point to the procedure division, which is placed in the PC when the task begins execution. The third word controls end action. When the value in the third word is greater than 15, the value is interpreted as the address of an end action routine, described in greater detail in paragraph 5.5. When the value is 15 or less, this value has no significance other than to indicate that no end action is to be taken.

In addition to dividing a task into data and procedure divisions, the user may make parts of the task into overlays, using the Link Editor. TX990 supports the automatic overlay loading capability of the Link Editor, as described in the *Model 990 Computer Link Editor Reference Manual*, part number 949617-9701.

Briefly, to create an overlayed program the user should assemble each intended overlay as a separate module, placing the output from the assemblies into separate files. All calls to overlay modules must be Branch and Load Workspace Pointer (BLWP) instructions, with the entry point of the overlay as the operand (no indexing or register operand is allowed). Next, the program must be linked, using the Link Editor. The user specifies the overlay structure of the problem through various link editor commands. The link editor LOAD directive must be used, to notify the Link Editor that automatic overlay loading is desired. See the *Link Editor Reference Manual* for more explanaiton of overlayed programs and usage of the Link Editor.

5.3 TASK SCHEDULING

The task scheduler uses a priority scheme with four levels and maintains a list of active tasks at each priority level. A task is added at the end of the active list in each of the following cases:

- When the task is placed in execution (bid).
- At the completion of a time slice.

A time slice is a period of execution of a task, beginning when the scheduler passes control to the task. A time slice ends when any of the following occurs:

- The system suspends the task upon expiration of the maximum time period allowed for a time slice.
- The task executes a supervisor call that suspends the task.
- The system suspends the task to await completion of an I/O operation.

The maximum time period allowed for a time slice is a system parameter specified when the system is generated.

When the currently executing task completes a time slice, the task scheduler passes control to the oldest task on the active list for the highest priority (0). If there is no task on the active list for priority 0, the oldest task on the active list for the next highest priority receives control.

To avoid completely locking out low priority tasks, there is a maximum number of consecutive time slices (weighting factor) for each priority level. When this number of time slices has been used by a priority level, the oldest task on the active list for the next lower priority is allowed a time slice before the higher level again has control. The maximum number of time slices for each priority level are system parameters defined during system generation.

Task management maintains a state code for each task. The state codes are listed in Appendix G.

5.4 PREVENTING ACCIDENTAL ALTERATION OR DESTRUCITON OF THE OPERATING SYSTEM

The most important consideration in programming user tasks is that the tasks do not interfere with the proper operation of the operating system. TX990 executes in either the Model 990/4 or the Model 990/10 Computer. Programming considerations to prevent accidental alteration or destruction of the operating system are different in each model.

In the Model 990/10 Computer, user tasks may execute in either the privileged or nonprivileged mode. An attempt to execute any of the following assembly instructions (described in the *Model 990 Computer TMS 9900 Microprocessor Assembly Language Programmer's Guide* part number 943441-9701) in the nonprivileged mode will result in a fatal error:

- RSET
- IDLE
- CKOF
- CKON
- LREX
- SBO, when the effective CRU address is $E00_{16}$ or greater
- SBZ, when the effective CRU address is $E00_{16}$ or greater
- TB, when the effective CRU address is $E00_{16}$ or greater

- LDCR, when the effective CRU address is $E00_{16}$ or greater
- STCR, when the effective CRU address is $E00_{16}$ or greater
- LIMI
- LMF (only available in computers having map option)
- LDS (only available in computers having map option)
- LDD (only available in computers having map option)

Although privileged and non-privileged modes are not used in the Model 990/4 Computer, generally, the user should avoid using the above instructions. If necessary the user may execute some of the above instructions with care; however, the system may operate improperly or fail to operate.

Execution of an RSET instruction has unpredictable results. Execution of an IDLE instruction places the computer in the Idle mode until an interrupt occurs. The CKOF and CKON instructions control the real-time clock, which is used by the system to allot execution times to tasks according to the scheduling algorithm. A CKOF instruction must not be executed, and a CKON instruction should not be required. Execution of an LREX instruction transfers control of the program that controls the programmer panel, stopping system operation. Execution of an LIMI instruction may interfere with the system control of interrupts. An LIMI instruction with an operand of 0 inhibits all interrupts. This instruction is permissible if the user reenables interrupts by executing an LIMI instruction having an operand of 15 as soon as possible. A user task should use the Do Not Suspend supervisor call to inhibit suspension of the task at the end of the current time slice rather than an LIMI instruction. Integrity of system time is lost if interrupts are not reenabled within 8.3 ms (60-Hz line frequency) or 10 ms (50-Hz line frequency).

The operating system uses the higher-order Communications Register Unit (CRU) addresses for dedicated purposes. None of the CRU instructions (SBO, SBZ, TB, LDCR, or STCR) should access any CRU addresses greater than DFF_{16} . For example, if workspace register 12 contained $1BF0_{16}$ as the CRU base address, an SBO instruction with a displacement of 8 results in an effective CRU address of $E00_{16}$, a dedicated address. Displacements less than 9 would result in valid addresses. The same base address in workspace register 12 would result in accessing dedicated addresses if an STCR instruction to store 8 or more bits were attempted. Refer to the *Model 990 Computer Assembly Language Programmer's Guide*, part number 943441-9701, for details of CRU addressing.

The user may write routines to perform extended operations (XOP). For tasks executing under TX990 in a Model 990/4 Computer, the same restrictions to the use of instructions apply both in the task and in the XOP routine. For tasks executing under TX990 in a Model 990/10 Computer, the restrictions that apply to the Model 990/4 Computer apply also in the XOP routine in the Model 990/10 Computer, because the Model 990/10 Computer is placed in the privileged mode by execution of an XOP instruction. Additional information about programming extended operations is included in Appendix C, entitled "User-Supplied Modules".

5.5 USER-SPECIFIED END ACTION ROUTINE IN RESPONSE TO FATAL ERRORS

The user specifies end action by placing the address of an end-action routine in the third word of the user task. When TX990 detects a fatal error and terminates the task abnormally, control transfers to the address in the third word unless that word initially contained a value of 15 or less. This allows the user to supply a routine to perform any required terminating function.

TX990 places the appropriate error code in the third word of the task after obtaining the end action routine address. After performing end action, or when the error is detected and no end action is specified, TX990 releases all I/O devices, closes all files, and removes the task from execution. The task error codes are listed in Appendix H, entitled "Printout of Fatal Task Error Codes and Illegal Interrupt Code".

When a task that has taken end action is to be reexecuted without reloading, the end-action routine must restore the end-action-routine address in the third word of the task.

5.6 CODING SUPERVISOR CALLS AND SUPERVISOR CALL BLOCKS

A user task requests support of the operating system by executing a supervisor call, XOP Level 15. A supervisor call block contains one or more bytes that define the supervisor call. The first byte of a supervisor call block contains the code of the supervisor call. Subsequent bytes are used as described for specific supervisor calls, when required.

The supervisor call may be coded in either of two ways. The first example shows a supervisor call coded as an XOP instruction:

```
XOP     @SCBA,15        Perform extended operation 15, by passing the
                        address corresponding to @SCBA to the system.
```

Alternatively, a symbol may be defined for supervisor calls using DXOP directive, as follows:

```
DXOP    SVC,15          Define symbol SVC for extended operation 15.
```

Supervisor calls following that directive may use the defined symbol, SVC, in the operation field and the address of the supervisor call block in the operand field as follows:

```
SVC     @SCBA           Perform extended operation 15 using the supervisor
                        call block at address @SCBA.
```

Supervisor call blocks are coded using BYTE directives, DATA directives, or both. Some supervisor call blocks must be aligned on word boundaries (i.e., they must have even addresses). Use of a DATA directive assures word alignment, but a BYTE directive does not perform word alignment. The descriptions of supervisor calls in the previous paragraphs identify the supervisor call blocks that must be word-aligned.

The following examples show coding for supervisor call blocks:

```
SCBA BYTE >10,0         Place 10(16) , the code for a Get Common Data Address
                        supervisor call, in a byte at location SCBA. The second
                        byte contains 0.

SCBB DATA >0300,DTBUF   Place 03(16) , the code for a Get Date and Time supervisor
                        call, in the first byte of a two-word block at location
                        SCBB, a word boundary. The second byte of the block
                        contains 0, and the last two bytes contain the address
                        corresponding to location DTBUF, a five-word buffer into
                        which the function places the date and time.
```

Alternatively, the label for a supervisor call block may be supplied with an EQU directive, as follows:

```
SCBB EQU $
     DATA <300,DTBUF
```

Assign label SCBB to current location. Place 03 in the first byte of a two-word block and the address corresponding to location DTBUF in the second word of the block, as in the preceding example.

The preceding example produces the intended result only if the current location is a word-aligned location. When the statements in the example follow a BYTE directive or a TEXT directive the location may not be word-aligned, and, if it is not, SCBB will not be the address of the supervisor call block. The following example will provide the desired result:

```
SCBB EVEN
     DATA <0300,DTBUF
```

If the location is not word-aligned, increment the location to a word boundary and assign label SCBB to the result. Otherwise, assign label SCBB to the location. Build the supervisor call blocks as in the preceding example.

SECTION VI

EXECUTIVE SUPERVISOR CALLS

6.1 INTRODUCTION

Executive supervisor calls (SVCs) in the TX990 operating system provide the user with the ability to control activities/operations in the user program, and to request services from the operating system. Table 6-1 shows the executive supervisor calls provided by TX990.

Table 6-1. Executive Supervisor Calls

SVC Name	Hexadecimal SVC Code
Bid Task	5
Change Priority	11
Do Not Suspend	9
Time Delay	2
Activate Time Delay	E
Unconditional Wait	6
Activate Suspended Task	7
End-of-Task	4
End-of-Program	16
Get Parameters	17
Get Own ID	20
Make Task Privileged	23
Convert Binary to Decimal	A
Convert Decimal ASCII to Binary	B
Convert Binary to Hexadecimal ASCII	C
Convert Hexadecimal ASCII to Binary	D
Get Memory	12
Release Memory	13
Get System Table	21
Get COMMON Data	10
Return COMMON Data	1B
Put Data	1C
Get Data	1D
Date and Time	3

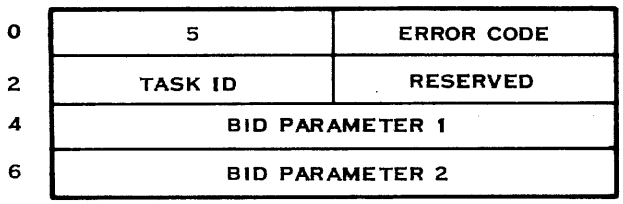
6.2 TASK CONTROL SUPERVISOR CALLS

Task control SVCs are used to schedule and control the execution of tasks. These SVCs are discussed in the following paragraphs.

6.2.1 BID TASK SUPERVISOR CALL 5_{16}. Bid Task supervisor call 5_{16} activates the specified task. The supervisor call block contains three, six, or eight bytes, aligned on a word boundary. Byte 0 contains the code, and the system returns a value in byte 1. Byte 2 contains the task identifier assigned to the task being bid. If the task is a linked-in task, the task ID was assigned during system generation. In systems supporting multiple dynamic tasks, the ID of a dynamic task is assigned by the IT OCP command. In single dynamic task systems, the task identifier of the task in the dynamic task area is 10_{16}. Byte 3 is unused. Bytes 4 through 7 may contain parameters to be passed to the task. The task obtains the parameters by executing a Get Parameters supervisor call.

When the system is unable to locate the specified task, it returns a -1 in byte 1 of the supervisor call block. Otherwise, the system returns the current task state code in that byte. The task state code for a terminated task is 4. Other task state codes are listed in Appendix G. When the task state code is not equal to 4, the supervisor call is ignored.

Bid task call block:



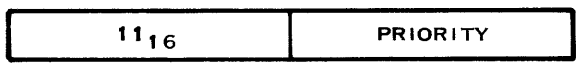
(A)137478

The following are examples or coding for supervisor call blocks for Bid Task calls:

SCBB DATA >0500,>1000,>4845,>4C50	Supervisor call block for a Bid Task call to bid task 10_{16}, and pass the ASCII representations of characters HELP to the task.
SCBA DATA >0500,>1A00	Supervisor call block for a Bid Task call to bid task $1A_{16}$. Undefined data from bytes corresponding to bytes 4 through 7 of the supervisor call block is passed to the task as parameters.

6.2.2 CHANGE PRIORITY SUPERVISOR CALL 11_{16}. Change priority supervisor call 11_{16} changes the priority of the calling task to a specified value. The supervisor call block contains two bytes and need not be aligned on a word boundary. Byte 0 contains the code, and byte 1 contains the new priority value. The valid priority levels are 0 through 3 (0 is usually reserved for system tasks). The system returns the previous priority value in byte 1. When the priority level is not 0 through 3, the previous priority level is not altered, and the system returns -1 in byte 1. The system returns the previous priority value in byte 1.

Change priority call block:



(A)137479

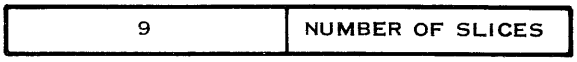
The following is an example of coding for a supervisor call block for a Change Priority call:

SCBP BYTE >11,1	Change the priority of the calling task to 1.

6.2.3 DO NOT SUSPEND SUPERVISOR CALL 9_{16}. Do Not Suspend supervisor call 9_{16} causes the system to override the time slice for the calling task by inhibiting the system from suspending the task. The task may suspend itself by executing an I/O supervisor call, or a Time Delay, Wait for I/O, or Unconditional Wait supervisor call. The supervisor call block contains two bytes, and need not be aligned on a word boundary. Byte 0 contains the code, and byte 1 contains 0 or a positive number. When byte 1 contains 0, the task will not be suspended for a system time unit. When byte 1 contains a number, the task will not be suspended for that number of system time units. Suspension of a task may be inhibited for a period from 1 to 255 system time units.

The length of a system time unit is dependent on the power-line frequency. For 60-Hz power, the system time unit is 50 ms. For 50-Hz power the system time unit is 40 ms.

Do not suspend call block



(A)137490

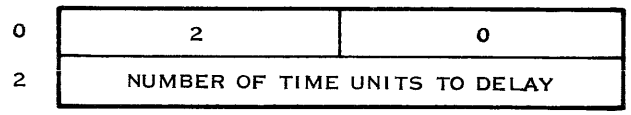
The following is an example of coding for a supervisor call block for a Do Not Suspend call:

```
SCBI BYTE 9,5          Inhibit suspending of calling task for five system time units.
```

The Do Not Suspend supervisor call should be used instead of an LIMI instruction having an operand of zero to inhibit suspension of a task at the end of the current time slice. When a task manipulates a data structure that is used by several tasks, the task should complete its alterations to the data structure before any of the tasks that use the data execute again. The Do Not Suspend supervisor call allows such a task to lock out other tasks while changing the data.

6.2.4 TIME DELAY SUPERVISOR CALL 2_{16}. Time Delay supervisor call 2_{16} suspends the calling task for a specified time period. The supervisor call block contains four bytes, aligned on a word boundary. Byte 0 contains the code, and byte 1 contains zero. Bytes 2 and 3 contain the number of system time units during which the task is to be suspended. The range of time-delay-suspension periods is from 1 to 32,767 system time units. When a negative number is used, the task is suspended for 1 system time unit. The system time unit is defined in the preceding paragraph.

Time delay call block:



(A)137481

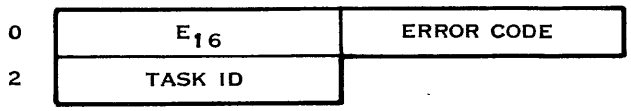
The following example shows coding for a supervisor call block for a Time Delay call:

```
SCBT DATA >200,40      Suspend the calling task for 40 system time units.
```

6.2.5 ACTIVATE TIME DELAY TASK SUPERVISOR CALL E_{16}. Activate Time Delay Task supervisor call E_{16} activates the specified task if it is in the time delay state. The supervisor call block contains three bytes, and need not be aligned on a word boundary. Byte 0 contains the code, and the system returns a value in byte 1. Byte 2 contains the task identifier assigned to the task.

When the system is unable to locate the task in the system, it returns -1 in byte 1 of the supervisor call block. Otherwise, the system returns the current task state code in that byte. The task state code for a task in time delay is 5. Other task state codes are defined in Appendix G.

Activate time delay call block:



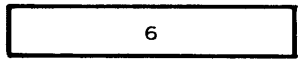
(A)137482

The following is an example of coding of a supervisor call block for an Active-Time-Delay Task call:

```
SCBR BYTE >E00,>13        Activate task 13(16), a task in a time delay.
```

6.2.6 UNCONDITIONAL WAIT SUPERVISOR CALL 6_{16}. Unconditional Wait supervisor call 6_{16} suspends the calling task indefinitely, unless an Activate Suspended Task supervisor call that specifies the task has already been executed. In that case, execution resumes immediately. Otherwise, the calling task remains suspended until another task executes an Active Suspended Task supervisor call for this task. The supervisor call block consists of a single byte that contains the code, and need not be aligned on a word boundary.

Unconditional Wait call block:



(A)137483

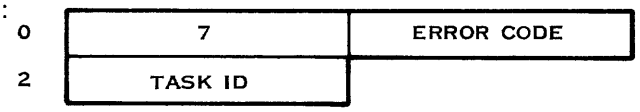
The following is an example of coding a supervisor call block for an Unconditional Wait call:

```
SCBW BYTE >6              Suspend the calling task unconditionally.
```

6.2.7 ACTIVATE SUSPENDED TASK SUPERVISOR CALL 7_{16}. Activate Suspended Task supervisor call 7_{16} activates the specified suspended task. When the specified task has not yet been suspended by an Unconditional Wait supervisor call, the effect of this supervisor call is to activate the task immediately following execution of an Unconditional Wait supervisor call. The supervisor call block consists of three bytes, and need not be aligned on a word boundary. Byte 0 contains the code, and the system returns a value in byte 1. Byte 2 contains the task identifier assigned when the task was loaded.

When the system is unable to locate the task in the system, it returns a -1 in byte 1 of the supervisor call block. Otherwise, the system returns the task state code in that byte. The task state code for a suspended task is 6_{16}. Other task state codes are defined in Appendix G.

Activate suspended task call block:



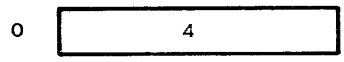
(A)137484

The following is an example of coding for a supervisor call block for an Active Suspended Task call:

```
SCBS BYTE  >7,0,>11        Activate task 11(16), previously suspended.
```

6. END OF TASK SUPERVISOR CALL 4_{16}. End of Task supervisor call 4_{16} closes all LUNOs assigned by the task and terminates the task. The supervisor call block consists of a single byte that contains the code, and need not be aligned on a word boundary.

End task call block:



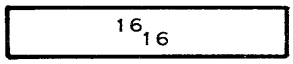
(A)137485

The following is an example of coding for a supervisor call block for an End-of-Task call:

```
SCBE BYTE    >4      Terminate calling task normally.
```

6.2.9 END OF PROGRAM SUPERVISOR CALL 16_{16}. End-of-Program supervisor call 16_{16} terminates a task. The supervisor call block consists of a single byte that contains the code, and need not be aligned on a word boundary. The function of an End-of-Program call is to close all LUNOs assigned by the task, to terminate the task, and to execute the rebid task which is normally the TXDS control program. The rebid task is discussed in Appendix J.

End program call block:



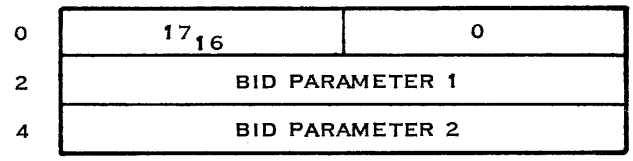
(A)137486

The following is an example of coding for a supervisor call block for an End of Program call:

```
SCBP BYTE    >16     Terminate calling task normally.
```

6.2.10 GET PARAMETERS SUPERVISOR CALL 17_{16}. Get Parameters supervisor call 17_{16} obtains task parameters passed to the task by an EXECUTE OCP command or by a Bid Task supervisor call. When a task includes a Get Parameters supervisor call, it must be the first supervisor call in the task. The supervisor call block consists of six bytes, aligned on a word boundary. Byte 0 contains the code, and byte 1 contains zero. The system places the parameters in bytes 2 through 5, in the same order in which they were placed in the EXECUTE command or in the Bid Task call.

Get parameters call block:



(A)137487

The following is an example of a supervisor call block for a Get Parameter call:

```
SCBG DATA  >1700,0,0    Obtain task parameters in two words at location SCBG+2.
```

6.2.11 GET OWN ID SUPERVISOR CALL 20_{16}. Get Own ID supervisor call 20_{16} returns the Task ID of the user task in byte 1 of the call block. The call block contains two bytes and need not be aligned on a word boundary. Byte 0 contains the code, and byte 2 is the task ID returned by the system.

Get own ID call block:

(A)137488

The following is an example of coding of the Get-Own-ID supervisor call:

```
SCBB BYTE    >20,0.
```

6.2.12 MAKE TASK PRIVILEGED SUPERVISOR CALL 23_{16}. This supervisor call will make the calling task privileged, which allows the task to perform direct disc I/O, and to execute a Get System Table supervisor call. The supervisor call block contains two bytes and need not be aligned on a word boundary. Byte 0 is the code and byte 1 is unused.

Make task privileged call block:

(A)137439

The following is an example of the Make Task Privileged supervisor call:

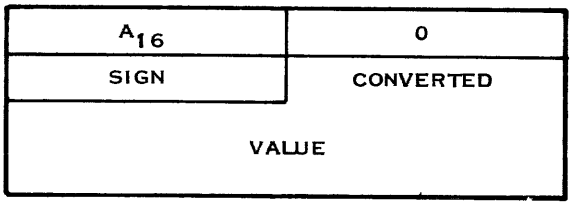
```
SCBB BYTE   >23,0
```

6.3 CODE CONVERSION SUPERVISOR CALLS

The code conversion group of supervisor calls consist of four supervisor calls that convert the binary value in a word to ASCII characters, or a group of ASCII characters to a binary value. The supervisor calls are described in the following paragraphs.

6.3.1 CONVERT BINARY TO DECIMAL ASCII SUPERVISOR CALL A_{16}. Convert Binary to Decimal ASCII supervisor call A_{16} converts the binary value in task workspace register 0 to the ASCII representation of the equivalent decimal value. The supervisor call block consists of eight bytes, and need not be aligned on a word boundary. Byte 0 contains the code and byte 1 contains zero. The system places an ASCII minus sign in byte 2 when the value being converted is negative, or a blank in that byte when the value is zero or positive. The converted value is placed in bytes 3 through 7, right-justified with leading blanks.

Convert bin to decimal ASCII call block:



(A)137490

The following is an example of coding a supervisor call block for a Convert Binary to Decimal ASCII call:

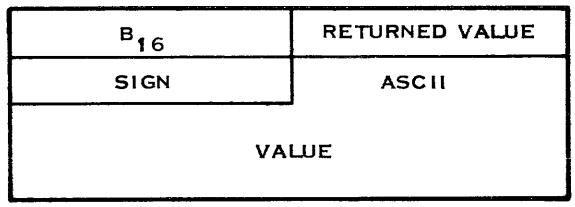
```
SCBD BYTE     >A,0,0,0,0,0,0,0      Convert value in workspace register 0 to decimal
                                    ASCII and place the result in the supervisor call
                                    block.
```

The following examples show typical values and the results:

Register 0	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0001_{16}	20_{16}	20_{16}	20_{16}	20_{16}	20_{16}	31_{16}
$7FFF_{16}$	20_{16}	33_{16}	32_{16}	37_{16}	36_{16}	37_{16}
$FFFF_{16}$	$2D_{16}$	20_{16}	20_{16}	20_{16}	20_{16}	31_{16}

6.3.2 **CONVERT DECIMAL ASCII TO BINARY SUPERVISOR CALL B_{16}.** Convert Decimal ASCII to Binary supervisor call B_{16} converts the ASCII characters in bytes 2 through 7 of the supervisor call block to a binary value, and places the value in task workspace register 0. The supervisor call block consists of eight bytes and need not be aligned on a word boundary. Byte 0 contains the code; and the system returns a value in byte 1. The calling task places the sign of the decimal value in byte 2, which may be the ASCII representation of +, -, zero, or blank. A - identifies the value as negative, and +, blank, or zero identifies the value as zero or positive. The ASCII representations of the decimal digits are entered in bytes 3 through 7, right-justified with leading blanks or ASCII zeros. When the system is able to perform the complete conversion correctly, it returns a zero in byte 1. When one or more of the characters are not valid, or the decimal number is not within the range of -32,768 to 32,767, the system is not able to complete the conversion, and returns a value of -1 in byte 1.

Convert decimal ASCII to binary call block:



The following example shows the coding of a supervisor call block for a Convert Decimal ASCII to Binary call:

```
SCBB BYTE >B,0,'0',' ',' ','3','7','8'
```

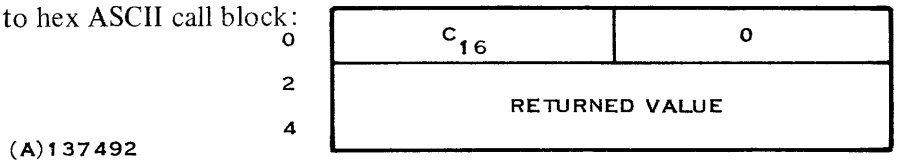
Convert 378 to its binary equivalent and place the result in workspace register 0.

The following examples show typical ASCII values and results.

Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Register 0	Byte 1
$2B_{16}$	20_{16}	20_{16}	20_{16}	20_{16}	31_{16}	0001_{16}	0
20_{16}	20_{16}	20_{16}	20_{16}	31_{16}	41_{16}	Undefined	FF_{16}
30_{16}	33_{16}	32_{16}	37_{16}	36_{16}	37_{16}	$7FFF_{16}$	0
$2D_{16}$	30_{16}	30_{16}	30_{16}	30_{16}	31_{16}	$FFFF_{16}$	0

6.3.3 **CONVERT BINARY TO HEXADECIMAL ASCII SUPERVISOR CALL C_{16}.** Convert Binary to Hexadecimal ASCII supervisor call C_{16}, converts the binary value in task workspace register 0 to the ASCII representation of the equivalent hexadecimal value. The supervisor call block consists of six bytes, and need not be aligned on a word boundary. Byte 0 contains the code, and byte 1 contains zero. The system returns the result in bytes 2 through 5.

Convert binary to hex ASCII call block:



The following is an example of coding of a supervisor call block for a Convert Binary to Hexadecimal ASCII call:

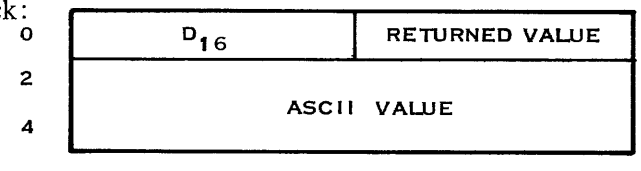
```
SCBH  BYTE  >C,0,0,0,0,0      Convert the value in workspace register 0 to hexadecimal
                              and place the result in the supervisor call block.
```

The following examples show typical values and the results:

Register 0	Byte 2	Byte 3	Byte 4	Byte 5
0001_{16}	30_{16}	30_{16}	30_{16}	31_{16}
$7FFF_{16}$	37_{16}	46_{16}	46_{16}	46_{16}
$FFFF_{16}$	46_{16}	46_{16}	46_{16}	46_{16}

6.3.4 CONVERT HEXADECIMAL ASCII TO BINARY SUPERVISOR CALL D_{16}. Convert Hexadecimal ASCII to Binary supervisor call D_{16} converts the ASCII characters in bytes 2 through 5 of the supervisor call block to a binary value, and places the value in task workspace register 0. The supervisor call block consists of six bytes, and need not be aligned on a word boundary. Byte 0 contains the code, and the system returns a value in byte 1. The calling task places the ASCII characters representing the hexadecimal digits in bytes 2 through 5. When the system is able to perform the complete conversion correctly, it returns a zero in byte 1. When one or more of the characters is not valid, the system is not able to complete the conversion, and returns a value of -1 in byte 1.

Convert hex ASCII to decimal call block:



(A)137493

The following is an example of coding a supervisor call block for a Convert Hexadecimal ASCII to Binary call:

```
SCBH  BYTE  >D,0,'0','3','F','4'      Convert 03F4₁₆ to binary.
```

The following examples show typical ASCII values and results:

Byte 2	Byte 3	Byte 4	Byte 5	Register 0	Byte 1
30_{16}	30_{16}	30_{16}	31_{16}	0001_{16}	0
30_{16}	30_{16}	33_{16}	$4B_{16}$	Undefined	FF_{16}
37_{16}	46_{16}	46_{16}	46_{16}	$7FFF_{16}$	0
46_{16}	46_{16}	46_{16}	46_{16}	$FFFF_{16}$	0

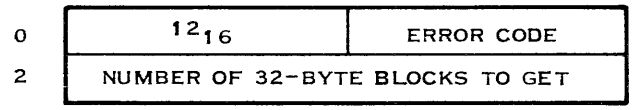
6.4 MEMORY ALLOCATION SUPERVISOR CALLS

The memory allocation group of supervisor calls consist of five supervisor calls that allow the user to access various blocks of memory. These supervisor calls are described in the following paragraphs.

6.4.1 GET MEMORY SUPERVISOR CALL 12_{16}. Get Memory supervisor call 12_{16} allocates a specified number of contiguous 32-byte blocks to the calling task. Each block is aligned on a word boundary. The address of the first byte in the first block is returned in user task workspace register 9. The supervisor call block consists of four bytes, aligned on a word boundary. Byte 0 contains the code, and the system returns a value in byte 1. Bytes 2 and 3 contain the number of blocks desired.

The TX990 Operating System allocates memory in the dynamic task space. If the requested memory is not available, TX990 returns a -1 in byte 1 of the supervisor call block.

Get memory call block:



(A)137494

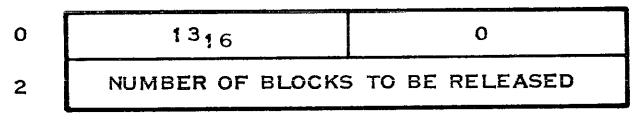
The following is an example of a supervisor call block for a Get Memory call:

```
SCBG DATA >1200,16
```

Allocate 256 words of memory to the calling task and return the address of the memory area in workspace register 9.

6.4.2 RELEASE MEMORY SUPERVISOR CALL 13_{16}. The Release Memory supervisor call 13_{16} returns memory to the available pool in the dynamic task area. The calling task places the address of the first byte of the first block of memory that would be released in task workspace register 9. The supervisor call block consists of four bytes, aligned on a word boundary. Byte 0 contains the code, and byte 1 contains zero. Bytes 2 and 3 contain the number of blocks that would be released.

Release memory call block:



(A)137495

The following is an example of a supervisor call block for a Release Memory call:

```
SCBR DATA >1300,16
```

Release 256 words of memory starting at the address specified in workspace register 9.

6.4.3 GET SYSTEM TABLE SUPERVISOR CALL 21_{16}. Get System Table supervisor call 21_{16} returns to the caller the address of the system table in which pointers to data structures within TX990 are located. This call can only be made by a privileged task. The supervisor call block contains four bytes, bytes 0 through 3, and must be aligned on a word boundary. Byte 0 contains the supervisor-call code; byte 1 is unused; and bytes 2 and 3 are returned by the system and contain the address of the system table.

The system table has the following format:

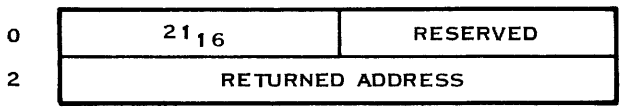
- Word 0 is a pointer to the system time and date block. The time and date blocks consist of five words with the following data:

  year, Julian day, hour, minute, and second.

- Word 1 is a pointer to the first element in the Task-Status-Block (TSB) chain.
- Word 2 is a pointer to the first element in the Physical-Device-Table (PDT) chain.
- Word 3 is a pointer to the first element in the Logical-Device-Table (LDT) chain.
- Word 4 is a pointer to the default disc name.

- Word 5 is a pointer to the Device Name Table (DNT).
- Word 6 is a pointer to the first element in the Procedure Status Block (PSB) chain.

Get system table call block:



(A)137496

Coding example:

```
SCBB    DATA    >2100,0
```

with the address of the system table returned in the second word of the call block.

6.4.4 GET COMMON DATA ADDRESS SUPERVISOR CALL 10_{16}. Get Common Data Address supervisor call 10_{16} causes the system to return the address of the beginning of the COMMON area of memory in task workspace register 9, and the size of the area (in bytes) in task workspace register 8. The size of the COMMON area is a system parameter specified at system generation. The supervisor call block consists of two bytes and need not be aligned on a word boundary. Byte 0 contains the code, and the system returns a value in byte 1.

When no intertask common has been specified for the system, the system returns a -1 in byte 1 of the supervisor call block. Otherwise, the system returns 0 in that byte.

Get COMMON call block:

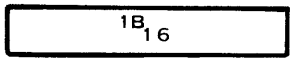
(A)137497

The following is an example of coding a supervisor call block for a Get Common Data Address supervisor call:

```
SCBG BYTE    >10,0      Supply address and size of common memory in workspace
                        register 9 and 8, respectively.
```

6.4.5 RETURN COMMON DATA SUPERVISOR CALL $1B_{16}$. Return Common Data supervisor call $1B_{16}$ performs no operation in TX990 Operating System. The call is included for compatibility with DX10 Operating System. The supervisor call block consists of single byte that contains the code, and need not be aligned on a word boundary.

Return COMMON call block:



(A)137498

The following is an example of the coding of a supervisor call block for a Return Common Data call:

```
SCBR BYTE    >1B        Perform a no-operation.
```

6.5 INTERTASK COMMUNICATION SUPERVISOR CALLS

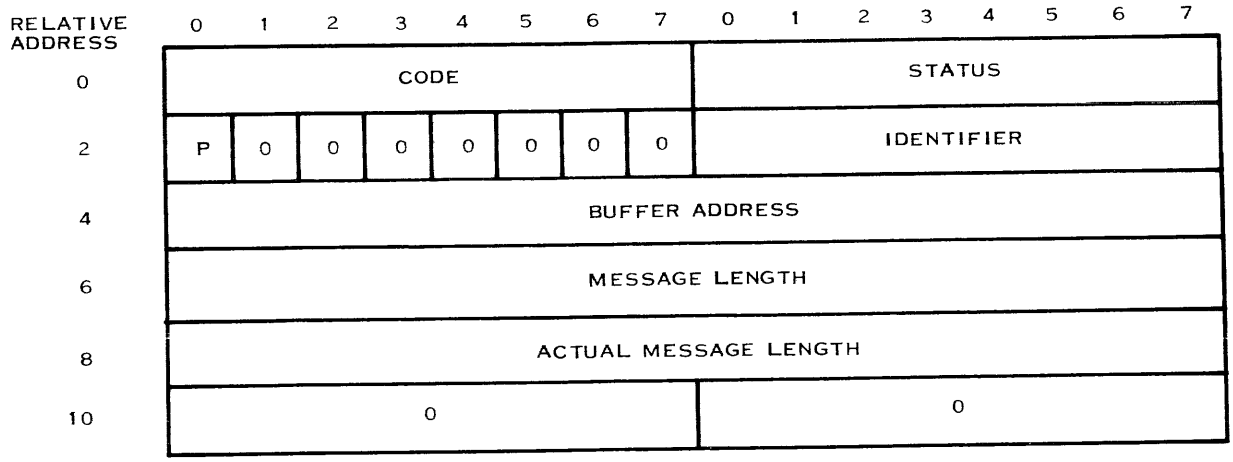
The Intertask Communication supervisor calls allow the user to pass messages between tasks. The system must include Intertask Communication and buffers for messages specified during system generation.

The Intertask Communication supervisor calls are the Put Data and the Get Data supervisor calls. Each call specifies an identifier that corresponds to a queue of data messages. Put Data calls that specify a given identifier place the data messages in the queue in the order in which the calls are executed. Get Data calls that specify the same identifier retrieve the data messages in the first-in first-out sequence.

Messages are queued in buffers supplied by Buffer Management. The number of buffer pools, the sizes of the buffers, and the numbers of buffers in each pool are specified during system generation. The number of characters required for a message buffer includes the overhead (eight characters per message). When a task executes a Put Data supervisor call to place an 80-character message in a buffer, the system requests a buffer at least 88 characters in size. Buffer pools must be specified to provide an adequate number of buffers of adequate size to support the Intertask Communication supervisor calls in the concurrently executing tasks.

6.5.1 PUT DATA SUPERVISOR CALL $1C_{16}$. Put Data supervisor call $1C_{16}$ places a message from a specified buffer in the user's task into a queue of data messages. The supervisor call block consists of 12 bytes aligned on a word boundary as shown in figure 6-1. Byte 0 contains the code and the system places a status code in byte 1 at the completion of the operation. Byte 2 contains the Purge flag in the most significant bit. The Purge flag should be set to zero for a Put Data supervisor call. Byte 3 contains the identifier, a number less than 255. Bytes 4 and 5 contain the address of the buffer that contains the message. Bytes 6 and 7 are not used by the Put Data supervisor call. Bytes 8 and 9 contain the number of characters in the message. Bytes 10 and 11 contain zero.

The operation places the specified number of characters from the specified buffer into the queue corresponding to the identifier. The system returns zero in byte 1 when the operation completes sucessfully, and -1 (FF_{16}) in byte 1 when memory is not available in a queue for the message.



(A) 137510

Figure 6-1. Intertask Communication Supervisor Call Block

The following is an example of coding for a supervisor call block for a Put Data supervisor call:

```
SCBP    DATA    >1C00                   Place message in buffer at address
                                        MSG1 in queue for identifier 15.
                                        Message contains 80 characters.
        BYTE    0       FLAGS
        BYTE    25      IDENTIFIER
        DATA    MSG1    BUFFER ADDRESS
        DATA    0       NOT USED
        DATA    80      80 CHARACTERS
        DATA    0       ZERO REQUIRED
```

6.5.2 GET DATA SUPERVISOR CALL $1D_{16}$. Get Data supervisor call $1D_{16}$ obtains a message from a specified queue and places it in the specified buffer in the user's task. The supervisor call block consists of 12 bytes aligned on a word boundary as shown in figure 6-1. Byte 0 contains the code and the system places a status code in byte 1 at completion of the operation.

Byte 2 contains the Purge flag in the most significant bit. When the Purge flag is set to 1, the supervisor call deletes all the messages in the specified queue and does not place a message in the specified buffer. When the Purge flag is set to 0, the supervisor call performs the normal operation. Byte 3 contains the identifier of the queue from which the message is obtained. Bytes 4 and 5 contain the address of the buffer into which the message is placed. Bytes 6 and 7 contain the maximum number of characters to be placed in the buffer. The system places the number of characters actually received in bytes 8 and 9 at completion of the operation. Bytes 10 and 11 contain zero.

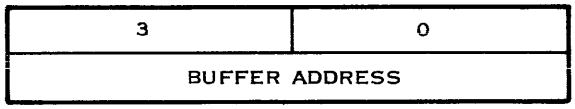
When the Purge flag is set to 0, the operation transfers a message from the queue associated with the specified identifier to the user's buffer. Messages in the queue are transferred in first-in first-out order. The number of characters transferred is the number of characters in the message or the number of characters specified in bytes 6 and 7 of the supervisor call block, whichever is less. When the Purge flag is set to one, the operation deletes all the messages in the queue associated with the specified identifier. The system returns zero as a status code in byte 1 when the operation completes successfully. When there is no message in the specified queue, the system returns -1 (FF_{16}) in byte 1. The following is an example of coding for a supervisor call block for a Get Data supervisor call:

```
SCBG    DATA    >1D00   GET DATA                Obtain a message up to 64 charac-
        BYTE    0       NORMAL OPERATION        ters in length (or the first 64 char-
        BYTE    35      IDENTIFIER              acters of a longer message) from the
        DATA    RMSG    BUFFER ADDRESS          queue for identifier 35 and place
        DATA    64      64 CHARACTERS           the message in a buffer at location
        DATA    0       ACTUAL MESSAGE LENGTH   RMSG.
        DATA    0       ZERO REQUIRED
```

6.6 DATE AND TIME SUPERVISOR CALL 3_{16}

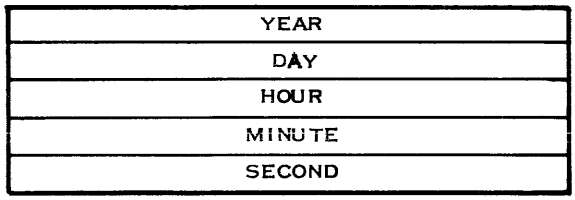
Date and Time supervisor call 3_{16} returns date and time in binary form. The supervisor call block consists of four bytes, aligned on a word boundary. Byte 0 contains the code, and byte 1 contains zero. Bytes 2 and 3 contain the addresses of a five-word area into which the function places the result. The binary values corresponding to the year, day, hour, minute, and second are placed in the first through fifth words of that area, respectively.

Date and time call block:



(A)137499

Buffer block:



(A)137500

The following is an example of coding for a supervisor call block for a Date and Time call:

```
SCBT DATA   >300,DAT
```

Place date and time data in a five-word area at location DAT.

SECTION VII

DEVICE AND FILE I/O SUPERVISOR CALLS

7.1 INTRODUCTION

The TX990 Operating System provides several supervisor calls used for performing I/O to devices and files. Supervisor call 00 may be used to perform many general I/O and file management operations which apply to many different types of devices and files. Three supervisor calls to perform character mode I/O to Video Display Terminals are provided. Also provided are a Wait for I/O SVC and two Abort I/O SVCs. Table 7-1 lists the I/O supervisor calls. The following paragraphs describe each SVC separately.

Table 7-1. I/O Supervisor Calls

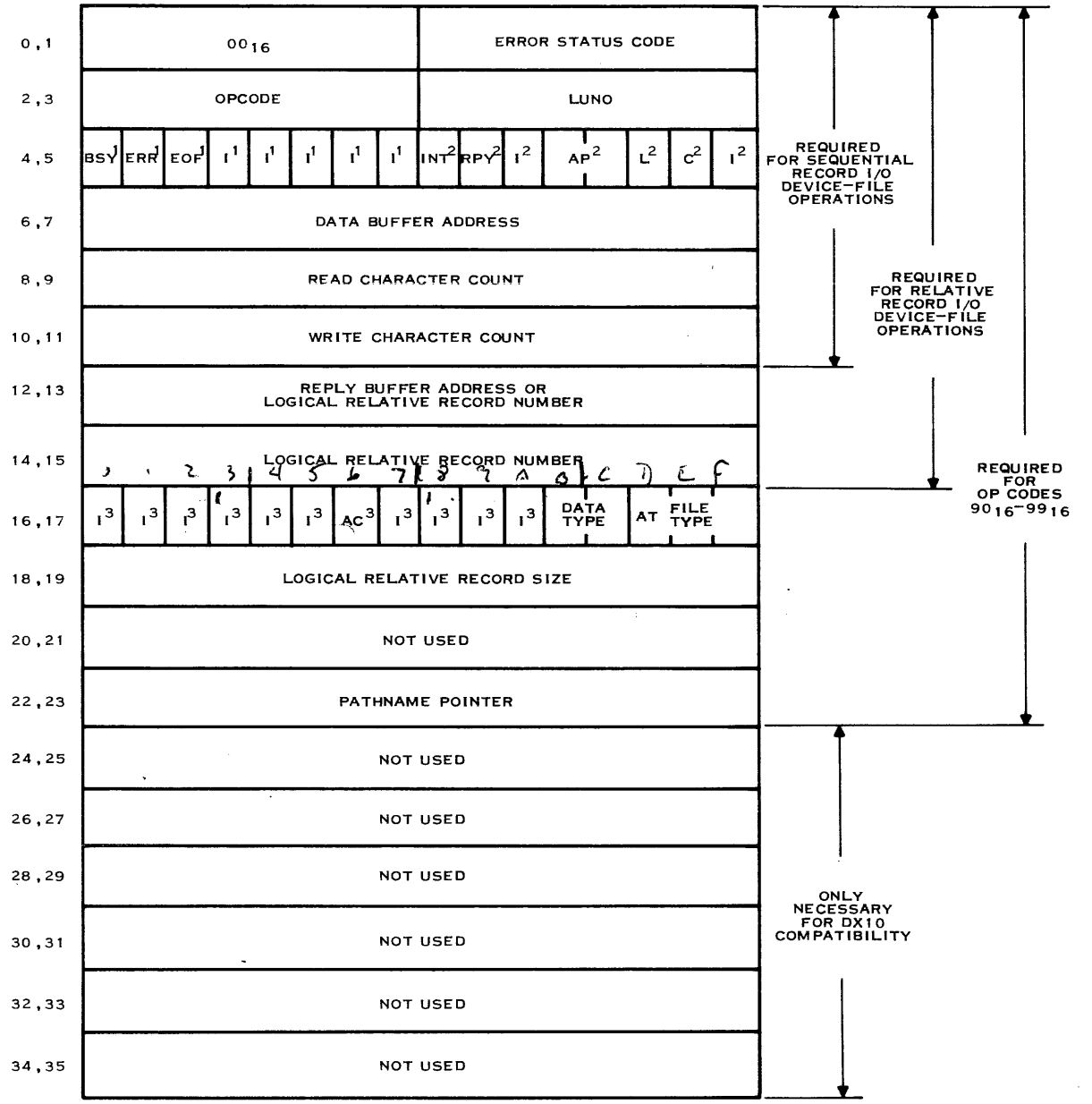
SVC Name	Hexadecimal SVC Code
General I/O	00
TX990 release 1.0 file management	15
VDT utility	1A
VDT character input	8
VDT conditional character input	18
Wait for I/O	1
Abort I/O by LUNO	F
Abort I/O by call block	1E

7.2 I/O SUPERVISOR CALL (00)

The general I/O supervisor call (SVC 00) is used to perform I/O to devices and files, and to perform many file manipulation operations. The SVC is programmed by coding the appropriate bytes in the call block, as shown in figure 7-1.

The coding of the bytes is described below.

- Byte 0 Coded by user with 00_{16} (for supervisor call 00_{16}).
- Byte 1 Error status code returned by TX990. For the list of error status codes, see Appendix I – I/O Error Codes.
- Byte 2 Coded by user with the I/O operations code (described in paragraph 7.2.1).
- Byte 3 Coded by user with the LUNO number to be associated with the I/O device or file to be utilized for the I/O operation. The LUNO is ignored during Create File, Delete File, Compress File, Unprotect File, Write Protect File, and Delete Protect File Operations.



NOTE 1: SYSTEM FLAGS
- BSY = BUSY FLAG
- ERR = ERROR FLAG
- EOF = END-OF-FILE FLAG
- I = IGNORED

NOTE 2: USER FLAGS
- INT = INTIATE FLAGS
- RPY = REPLY FLAG
- AP = ACCESS PRIVILEGE FLAG
- L = LOCK AND UNLOCK FLAG
- C = COMMUNICATION TIME-OUT FLAG
- I = IGNORED

NOTE 3: UTILITY FLAGS
- AC = AUTO-CREATE FLAG
- I = IGNORED
- AT = ALLOCATION TYPE (CONTIGUOUS/ NONCONTIGUOUS)

(B)135908B

Figure 7-1. Supervisor Call Block for I/O Supervisor Call 00_{16}

NOTE

Bytes 4 through 15 are ignored by I/O operation codes 90_{16} through 99_{16}.

- Byte 4, bit 0 — TX990 places a 1 in the busy bit during an I/O operation. When the task codes the Initiate I/O flag to 1, it must monitor this bit when waiting for the I/O to complete.

- Byte 4, bit 1 — TX990 places a 1 in the error bit when the I/O operation terminates in error. It is 0 for normal termination.

- Byte 4, bit 2 — The operating system sets this bit to 1 when it detects an end-of-file record; otherwise, it is set to 0.

- Byte 4, bit 3-7 — Unused.

- Byte 5, bit 0 — Coded by the user with the initiate I/O flag. When this bit is set to 1, the operating system will begin the specified I/O operation and return control to the task before the I/O is completed. When this bit is set to 0, the operating system will not return control to the task until the I/O operation is completed. This bit is useful when buffering I/O.

- Byte 5, bit 1 — Coded by the user with the reply flag. This bit should only be set to 1 when the I/O operation is a write and the LUNO specifies an interactive device. When the bit is set to 1, the specified write will be followed by a read, allowing the task to receive a reply from the interactive device. When this bit is 1, byte 12-13 must contain the address of a reply block. A reply block is a 3-word block: the first word is coded by the user and must contain the buffer address that contains the characters entered in reply to the write operation; the second word must contain the maximum number of characters expected to be entered (usually the size of the reply buffer); the third word will be returned by the operating system and will contain the number of characters actually entered in reply to the write operation.

- Byte 5, bit 2 — Ignored by TX990.

- Byte 5, bit 3-4 — Coded by the user to attain the access privilege, during an open operation. These bits are ignored during another operation.

  00 – Exclusive all flag indicating that only one task may open the file concurrently.

  10 – Shared flag indicates more than one task may open the file concurrently. This bit should only be set when doing an open operation to a relative record file.

- Byte 5, bit 5 Coded by the user to lock or unlock a logical relative record. When set, this flag specifies that the logical relative record is to be locked for all read operations and unlocked for all rewrite operations. A locked record cannot be read or rewritten by any task except the locking task, and then the locking task must use the same LUNO. Any task can unlock a record using a $4A_{16}$ I/O operation. When this flag is zero, then current status (locked or unlocked) is unchanged.

- Byte 5, bit 6 Coded by the user with the communications timeout flag. See the Communications manual for more detail.

- Byte 5, bit 7 Not used by TX990.

- Bytes 6-7 The user must code these bytes to contain the starting address of the I/O buffer.

- Bytes 8-9 The user must code these bytes when using a read I/O operation code. The bytes must contain the maximum number of characters to be read. When the record length specified exceeds the maximum number of characters that the device-file supports, only the number of characters that are supported by the device-file applies. Not used for write operations.

- Bytes 10-11 When a Write I/O operation is used, bytes 10-11 will contain the number of characters to be written. When the character count exceeds the maximum number of characters for the device-file, the maximum number of characters for the device-file applies. When a Read I/O operation is used, the operating system will return the actual number of characters read in these bytes.

- Bytes 12-13 When the user sets the reply flag to 1, these bytes must contain the address of the reply block described above. These bytes need not be appended to the above bytes if the I/O operations are directed to sequential device-files.

- Bytes 12-15 When the user is doing I/O operations to relative record files, these bytes contain the logical record number. The operating system will update these bytes after I/O operations. These bytes need not be appended to the above bytes, except when directing I/O to a relative record file.

- Bytes 16-34 Need not be appended to the above bytes except when using I/O operation codes 90_{16} through 99_{16}.

- Bytes 16-17 Utility Flags.

- Bytes 16-17, bit 0-2, 4-5, 7, 9-10 Need to be zero for DX10 compatibility. TX990 ignores it.

- Bytes 16-17, bit 3, 8 Need to be one for DX10 compatibility. TX990 ignores it.

- Bytes 16-17, bit 6 — User sets to one to request automatic logical file creation. It is set during an assign I/O device-file operation and causes a file to be created during an Open I/O operation.

- Bytes 16-17, bit 11-12 — The user must code these bits in the following manner:

Code	Meaning
00	Binary data
01	ASCII data
10,11	Reserved for new data formats

These bits are ignored except during create file operations.

- Bytes 16-17, bit 13 — Set to one when creating a noncontiguous file or to a zero when creating a contiguous file. TX990 will ignore the bit if it is set to zero and will create a noncontiguous file without flagging an error. TX990 supports only non-contiguous files. This bit is ignored except during create file operations.

- Bytes 16-17, bit 14-15 — Set to one of the following codes to specify the file type when the file is created. This field is ignored except for create operations.

Code	Meaning
00	undefined
01	sequential file
10	relative record file
11	key index file (not supported in TX990)

- Bytes 18-19 — When using create file operations to create a relative record file, the user must code these bytes with the number of characters in each record (i.e., record size). These bytes are ignored when creating a sequential file.

- Bytes 20-21 — These bytes are not used by TX990.

- Bytes 22-23 — The user must code these two bytes with address of a pathname. These bytes are used for Assign LUNO, Create File, Delete File, Change the Name, Unprotect File, Write Protect File, Delete Protect I/O operations. All other operations ignore this field. The pathname is in the following format:

Count	Pathname
Byte n	Byte 1-n

The first byte at the pathname address is the character count. The following bytes are the pathname itself.

- Byte 24-35 — Need only be appended for DX10 compatibility.

7.2.1. **I/O OPERATIONS.** The I/O SVC may be used to perform all of the operations listed in Table 7-2, by coding the correct I/O opcode in byte 2 of the call block, and correctly defining the remaining fields in the call block. The opcode for each operation is given in the table.

Table 7-2. SVC 00 I/O Operations

Operation	Hexadecimal Opcode
Open	00
Close	01
Close with end-of-file	02
Open with rewind	03
Close and unload	04
Read device status	05
Forward space	06
Backward space	07
Read ASCII	09
Read direct	0A
Write ASCII	0B
Write direct	0C
Write end-of-file	0D
Rewind	0E
Unload	0F
Create file	90
Assign LUNO to pathname	91
Delete file	92
Release LUNO assignment	93
Compress file	94
Assign new file name	95
Unprotect file	96
Write protect file	97
Delete protect file	98
Verify pathname	99
Unlock	4A

NOTE

Most of the "ninety" opcodes do not apply to device operations (exceptions are Assign and Release LUNO). Table 7-3 shows which of the other operations apply to which devices.

The following paragraphs describe each operation in detail.

7.2.2 **OPEN OPERATION (CODE 00_{16}).** The Open operation is specified by placing code 00_{16} in byte 2 of supervisor call block. A description of how the Open operation controls the I/O device and the file on the medium of the I/O device is presented in the following subparagraphs.

7.2.2.1 **Controlling the I/O Device.** When a device is designated during system generation to be in the file mode (i.e., available to be utilized by only one program), the open operation causes the device to be assigned solely to the calling program until a close operation (i.e., Close, Close with EOF, or Close Unload is executed. In addition, when a device is designated during system generation to be in the file mode, an open operation (i.e., Open or Open Rewind) must be performed before any other file management operation is executed. Until an open operation is executed, files on diskettes are not located and file management operations to manipulate them cannot be initiated or performed.

The following device operation(s) result from execution of an open operation:

- Dummy — No operation.
- 911 or 913 VDT — Performs a line feed and a carriage return.
- 733 ASR Keyboard/Printer — Performs a line feed and a carriage return.
- 33 ASR Teletype Keyboard/Printer — Performs a line feed and a carriage return.
- 33 ASR Teletype Paper Tape Punch, and Reader — Two bells are output and a RUB OUT key must be entered.
- Other devices — No operation.

When a device responds to an open operation, the operating system returns the device code number (table 7-3) to bytes 6 and 7 of the supervisor call block, which bytes are usually used for data buffer address field.

7.2.2.2 **Controlling the Diskette File.** When a relative record file is to be exclusively accessed, the shared access flag in byte 5 of the supervisor call block, for the open operation, must be set to zero. Sequential files are always exclusively accessed. The file then remains assigned to the program until the program is terminated or until a close operation is executed. In response to an open operation to a file, the operating system returns the file code number (table 7-4) to bytes 6 and 7 of the supervisor call block (which bytes are usually used for the data buffer address), code $01FF_{16}$ to indicate that a sequential file has been opened or code $02FF_{16}$ to indicate that a relative record file has been opened. When the auto create flag is set during the assign LUNO to pathname operation, the open operation causes a file to be created if the file does not already exist. When a relative record file is opened, the record size is returned in bytes 8-9 of the call block.

7.2.3 **CLOSE OPERATION (CODE 01_{16}).** The close operation is specified by placing code 01_{16} in byte 2 of File Management Supervisor Call Block 00_{16}. When specified by the user, the close operation releases the I/O device and the file on the medium of the I/O device from the calling task. When a task assigns more than one LUNO to an I/O device and opens them, a close operation should be executed for all LUNOs for which an open operation was performed. When a close operation is executed to a diskette file, the file is released from the task and any records locked by the task are released. When I/O device is designated to be in the file mode during system generation, a close operation (or close with EOF or close unload) must be executed to release the file from the calling task.

Table 7-3. I/O Operations (Record Mode)

I/O Code/Operation (Hexadecimal)	733 ASR/KSR or 743 KSR Keyboard/Printer	Card Reader	Line Printer	733 ASR Cassette Units	913/911 VDT (Record Mode)	Dummy Device	33 ASR Keyboard/Printer	33 ASR Punch	33 ASR Reader
00_{16} OPEN	R	R	R	R	R	R	R	R	R
01_{16} CLOSE	R	R	R	R	R	I	R	R	R
02_{16} CLOSE-WRITE EOF	R	E	R	R	R	I	R	R	R
03_{16} OPEN-REWIND	R	R	R	R	R	R	R	R	R
04_{16} CLOSE-UNLOAD	R	R	R	R	R	I	R	R	R
05_{16} READ DEVICE STATUS	I	I	I	I	I	I	R	R	R
06_{16} FORWARD SPACE	I	I	I	R	I	I	I	I	I
07_{16} BACKWARD SPACE	I	I	I	R	I	I	I	I	I
09_{16} READ ASCII	R	R	E	R	R	R	R	E	R
$0A_{16}$ READ DIRECT	E	R	E	R	E	I	E	E	R
$0B_{16}$ WRITE ASCII	R	E	R	R	R	I	R	R	E
$0C_{16}$ WRITE DIRECT	E	E	R	R	E	I	E	R	E

Table 7-3. I/O Operations (Record Mode)(Continued)

I/O Code/Operation (Hexadecimal)	733 ASR/KSR or 743 KSR Keyboard/Printer	Card Reader	Line Printer	733 ASR Cassette Units	913/911 VDT (Record Mode)	Dummy Device	33 ASR Keyboard/Printer	33 ASR Punch	33 ASR Reader
$0D_{16}$ WRITE EOF	R	E	R	R	R	I	R	R	E
$0E_{16}$ REWIND	I	I	R	R	I	I	I	I	I
$0F_{16}$ UNLOAD	I	I	I	R	I	I	I	I	I
$4A_{16}$ UNLOCK									

Note: Unlock I/O operation used for relative record diskette files.

R Response
I Ignored
E Error

Table 7-4. Device Code Numbers and File Code Numbers

Device/File Name	Device Code Number	File Code Number
Dummy	0000	
743 KSR or 733 ASR/KSR Keyboard/Printer	0001	
Line Printer	0002	
733 ASR Cassette Unit	0003	
Card Reader	0004	
911 or 913 Video Display Terminal	0005	
33 ASR Keyboard/Printer	0001	
33 ASR Paper Tape Punch	0003	
33 ASR Paper Tape Reader	0003	
Diskette	0006	
Communication Device	0007	
Sequential Record File		$01FF_{16}$
Relative Record File		$02FF_{16}$

No physical device operation(s) results from execution of a close operation.

7.2.4 **CLOSE WITH EOF OPERATION (CODE 02_{16}).** The close with EOF operation is specified by I/O operation code 02_{16}. The operation consists of the close operation previously described and a write EOF operation for the specified device. When the device specified is the keyboard/printer, the printer performs three line-feed operations. When the device specified is the line printer, the printer performs a form-feed operation. When the device specified is the 911 or 913 VDT, only the close operation is performed. The close with EOF operation is an illegal operation for the card reader and the system returns the appropriate error status code. When the device specified is the paper-tape punch, an ASCII EOF is punched (DC3, CR, LF, DC3, null, null, null, null) and 80 null frames are punched for trailer. If a close with EOF operation is executed to a diskette file, the current position of the file is stored as the end-of-file. The close with EOF operation for relative record writes the end-of-file to the record number in bytes 12 through 15 of the SCB.

7.2.5 **OPEN REWIND OPERATION (CODE 03_{16}).** The open rewind operation is specified by I/O operation code 03_{16}. The operation consists of an open operation previously described, and a rewind operation. When the device is the line printer, the device performs a formfeed operation. When the device is a VDT, the system blanks the screen and positions the cursor on column 0 of the last row, the home position. When the device is a cassette unit, the device rewinds the cassette and places it in the ready state. When the device is the keyboard/printer or the card reader, only the open operation is performed.

When the paper-tape punch is opened with rewind, two bells are output and the user then turns on the punch and presses the RUB OUT key. When the RUB OUT key has been pressed then 80 null frames are punched for leader. For a sequential diskette file, the current record is the first record in the file, and for a relative record file, the record number in the SCB is set to zero.

7.2.6 **CLOSE UNLOAD OPERATION (CODE 04_{16}).** The close unload operation is specified by I/O operation code 04_{16}. This operation consists of a close operation previously described, and an unload operation. When the device is the line printer, the device performs a form-feed operation. When the device is a cassette unit, the device rewinds the cassette to the clear area at the beginning of the tape. When the device is a keyboard/printer or the card reader, only the close operation is performed.

When the device is the 33 ASR paper-tape punch, 80 null frames are punched. A close unload operation to a diskette file causes the file to be closed.

7.2.7 READ DEVICE FILE STATUS OPERATION (CODE 05_{16}). The read device-file status operation is specified by I/O operation code 05_{16}. The operation is ignored by all devices except 33 ASR and diskette files.

When the device is the 33 ASR, the read character count (bytes 8 and 9) is cleared.

When the device-file is a diskette file, file characteristics and format are stored in the user's buffer address, which is specified by bytes 6, 7 of the supervisor call block. The buffer must contain 3 words. The operating system will return a 0 in word 1. Word 2 will contain the logical record length when file is a relative record file; otherwise it will contain a zero. Word 3 will contain the physical record length and will be 128.

7.2.8 FORWARD SPACE OPERATION (CODE 06_{16}). The forward space operation is specified by I/O operation code 06_{16}. The operation is ignored by all devices except the 733 ASR cassette units. The operation moves the cassette tape forward a specified number of records or until an end-of-file record is read. The number of records to be read is placed in bytes 10 and 11 of the supervisor call block. When an end-of-file record is read, the tape is positioned at the beginning of the record following the end-of-file. The end-of-file record consists of a DC3 (X-OFF) in the first character position of the record.

The forward space operation causes a diskette file to skip records in the forward direction. Bytes 10 and 11 of the Supervisor Call Block (SCB) contain the number of records to be skipped. For a relative record file, file management updates the record number in the SCB. If an end-of-file occurs before the specified number of records has been skipped, the end-of-file flag in the SCB is set, and the operation stops. The next operation accesses the first record following the end-of-file. Following the operation, bytes 10 and 11 of the supervisor call block contain the number of records remaining to be skipped; zero when an end-of-file did not occur.

7.2.9 BACKWARD SPACE OPERATION (CODE 07_{16}). The backward space operation is specified by I/O operation code 07_{16}. The operation is ignored by all devices except the 733 ASR cassette units. The operation moves the cassette tape a specified number of records in the reverse direction. Bytes 10 and 11 of the Supervisor Call Block (SCB) contain the number of records to be moved. The backward space operation causes a diskette file to skip records in the reverse direction. The number of records to be skipped is placed in bytes 10 and 11 of the SCB. For a relative record file, file management updates the record number in the SCB. For a sequential file, if an end-of-file occurs before the specified number of records has been skipped, the operation stops. The next operation accesses the first record after the EOF. Following the operation, bytes 10 and 11 of the SCB contain the number of records remaining to be skipped; zero when an end-of-file did not occur.

7.2.10 READ ASCII OPERATION (CODE 09_{16}). The read ASCII operation, code 09_{16}, reads a record of the specified file and stores the data, packed two characters per word in the buffer at the address in bytes 6 and 7 of the Supervisor Call Block (SCB). The maximum number of characters in the buffer is placed in bytes 8 and 9 of the SCB. The actual number of characters stored is placed in bytes 10 and 11 of the SCB. This number will be the number of characters in the record or the value in bytes 8 and 9, whichever is less. If an end-of-file occurs, file management sets the EOF bit in the SCB and sets the character count in bytes 10 and 11 of the SCB to zero. When the file is a relative record file and no end-of-file was encountered and the read operation is successful, file management increments the record number in bytes 12 through 15 of the SCB.

NOTE

The ASCII characters listed for each device in Appendix B are the valid characters for that device, and are stored with the most significant bit set to zero. Except as noted for the card reader, other characters are ignored.

When the device specified is either the keyboard/printer or the VDT, the operation sounds a tone to request the user to enter the characters at the keyboard. Characters are transferred to a buffer until the user enters a carriage return (New Line on a 913 VDT) or the number of characters specified in bytes 8-9 are entered. A carriage return is not included in the character count. If only a carriage return is entered, a zero is returned in the character count word in the SCB. When the device is the keyboard/printer, the user may correct the most recently-entered character by entering a backspace (CTRL/H). The character is deleted, and the printer performs a backspace and a line feed operation. When the device is a VDT, the most-recently entered character may be corrected by entering the left arrow (←), which deletes the character in the buffer and backspaces the cursor.

When the device specified is a 733 ASR cassette unit, the operation transfers characters from the cassette to the specified buffer until an end-of-record is detected or the number of characters specified in bytes 8-9 are read. The maximum number of characters in a cassette record is 83. When the number of characters to be read is less than the number of characters in the record, the remaining characters in the record are not available. An end-of-file record on the cassette is a record having a DC3 (X-OFF) character as the first character. The cassette unit does not provide a logical end-of-medium indication, but does provide a physical end-of-tape indication. The physical end-of-tape indication may indicate either end of the tape.

When the device specified is a card reader, the operation reads a card and transfers the characters read to the specified buffer. The number of characters specified is transferred up to a maximum of 80 characters. When the number of characters specified is greater than 80, only 80 characters are read and transferred. When fewer than 80 characters are specified, the remaining characters on the card are not available. Characters other than those listed in Appendix B for the card reader are placed in the buffer as spaces, and the system returns an error status code when these characters are read. The end-of-file record for the card reader has a slash in the first column and an asterisk in the second column (/*).

The read ASCII operation is an illegal operation for the line printer, and the system returns an error status code.

When the device specified is a 33 ASR paper-tape reader, the operation transfers characters from the paper tape to the specified buffer until an end-of-record is detected or the number of characters specified in bytes 8-9 are read. An end-of-file record on a paper tape is a DC3 (X-OFF, 13_{16}) character as the first character of the input record. The read ASCII operation for the 33 ASR paper-tape punch is an illegal operation and the system returns an error code.

7.2.11 READ DIRECT OPERATION (CODE $0A_{16}$). The read direct operation is specified by I/O operation code $0A_{16}$. The operation reads the number of characters specified as the record length in bytes 8 and 9 of the Supervisor Call Block (SCB). The operation transfers the characters to the buffer at the address in bytes 6 and 7 of the SCB without any characters translation and places the number of characters read in bytes 10 and 11 of the SCB.

When the device specified is the cassette unit, the system sets the most significant bit of each character to zero and stores the characters two per word. The read operation terminates when a complete physical record has been read.

When the device specified is the card reader, a column on the card is stored in a word of the buffer. The four most significant bits of the word are set to zero; the holes in the card are stored as ones, in the order shown in figure 7-2. The entire record is transferred to the buffer, and the end-of-file record is underfined for a read direct operation.

When the device specified is the 33 ASR paper-tape reader, each frame contains one byte of information in ASCII format.

A diskette file read direct operation is identical to the read ASCII operation for a diskette file.

When a read direct operation is attempted on the other devices, the system returns an error status code.

7.2.12 WRITE ASCII OPERATION (CODE $0B_{16}$). The write direct operation, code $0B_{16}$, transfers the data in the buffer at the address in bytes 6 and 7 of the Supervisor Call Block (SCB) to the specified file. The characters in the buffer are packed two per word. Bytes 10 and 11 of the SCB contain the number of characters to be written. When the file is a relative record file and the write operation is successful, file management increments the record number in bytes 12 through 15 of the SCB. A write ASCII operation clears any end-of-file indication for the current record or for a subsequent record.

NOTE

Each device recognizes the characters listed for the device in Appendix B. Other characters are ignored.

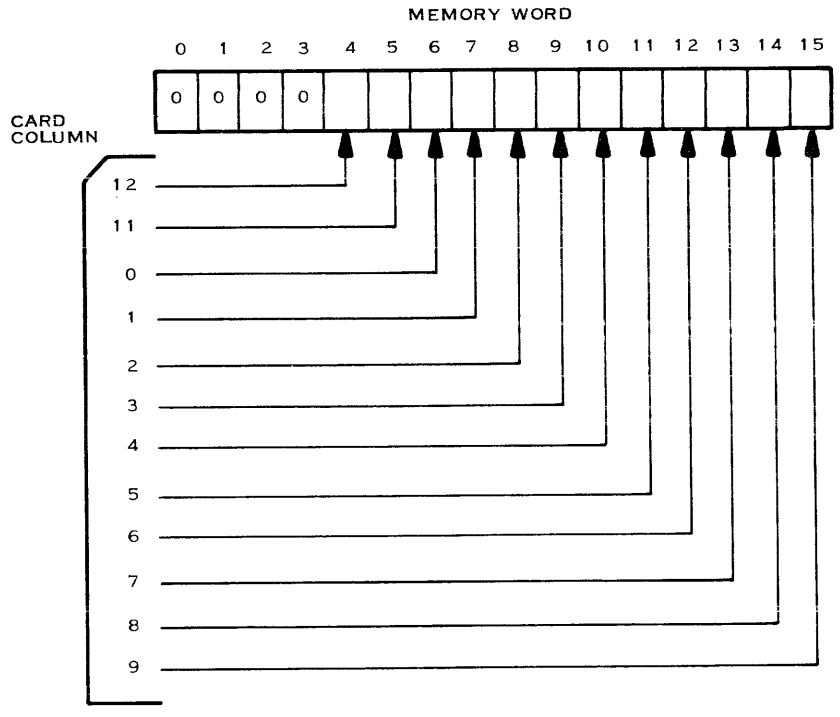


Figure 7-2. Bit Manipulation for Direct Read of Card

When the device specified is the keyboard/printer, the characters are printed. An HT character results in a space, and an FF character results in eight line feed operations. When the device specified is the line printer, the characters are also printed. Model 306 and Model 588 Line Printers recognize the SO($0E_{16}$) character as specifying a line of elongated characters. The SO character should be the first character of the line, and causes the printer to print double-width characters on the entire line. The number of characters per line is one-half the normal number of characters; i.e., 40 characters per 80-character line, or 66 characters per 132-character line.

When the device specified is a cassette unit, the characters are written on the cassette. A carriage return in the buffer is translated to an ETB (17_{16}) character. The maximum number of characters to be written is 83. When the buffer has been written, the system then writes a carriage return, a line feed, a DC4 (record off) character, and a DEL (rub out) character to indicate an end of physical record. When the reply bit is set, the response is read from the other cassette unit.

When a write ASCII operation is attempted on the card reader, the system returns an error status word.

When the device specified is the 33 ASR paper-tape punch, characters are written a byte at a time in the ASCII format. A carriage return in the buffer is translated to an ETB (17_{16}) character. The system punches an ASCII end-of-record (CR, LF, DC3, null, null, null, null) when the buffer is exhausted. When a write ASCII operation is attempted on the 33 ASR paper-tape reader, the system returns an error status word.

7.2.13 WRITE DIRECT OPERATION (CODE $0C_{16}$). The write direct operation is specified by the I/O operation code $0C_{16}$. The operation writes a record without performing any translation, writing the characters from the buffer at the address in bytes 6 and 7 of the Supervisor Call Block (SCB). The number of characters to be written is specified in bytes 10 and 11 of the SCB.

When the device specified is the cassette unit, the seven least significant bits of each byte are written on the cassette. The maximum number of characters per record is 83. If a DC4 character is embedded in the buffer, the system writes the character and also a DC2 (record on) to continue the operation. To assure that the last record is actually written on the tape, the user task should place a carriage return in the buffer. When the reply bit is set, the response is read on the other cassette unit.

When the device specified is the 33 ASR paper-tape punch, all eight bits of each byte are punched on the paper tape. If a DC4 (14_{16}) character is embedded in the buffer, the system punches the character and also a DC2 (record on, 12_{16}) to continue the operation. The system punches an end-of-record (DC3, null, null, null) when the buffer is exhausted.

When the device specified is a line printer, the seven least significant bits are sent to the printer and the installed options will determine how the characters are interpreted.

A diskette file write direct operation is identical to the write ASCII operation for a file.

When the write direct operation is attempted on any other device, the system returns an error status code.

7.2.14 WRITE EOF OPERATION (CODE $0D_{16}$). The write EOF operation is specified by I/O operation code $0D_{16}$. The operation consists of writing the end-of-file record defined for the specified device.

When the device specified is the cassette unit, the operation writes a DC3 (X-OFF) character on the cassette. When the device is the keyboard/printer, the operation performs three line-feed operations. When the device is the line printer, the operation performs a form-feed operation. When the device is the VDT, the operation is ignored. When the device is the card reader, the system returns an error status code, and no operation is performed.

When the device specified is a 33 ASR paper-tape punch, an ASCII end-of-file is punched.

When the unit specified is a diskette file, the write EOF operation writes the end-of-file record. There is no limit to the number of end-of-file records that may be written to a sequential file but there may be only one for relative record files. For a relative record file the end-of-file is written in the record specified (bytes 12-15 of the supervisor call block).

7.2.15 REWIND OPERATION (CODE $0E_{16}$). The rewind operation is specified by I/O operation code $0E_{16}$. When the device specified is the cassette unit, the operation rewinds the cassette tape to the clear area at the beginning of the tape, and then moves the tape in the forward direction to the beginning of tape marker, illuminating the READY indicator on the 733 ASR. When the specified device is the line printer, the operation performs a form feed operation.

A diskette file rewind operation simulates the rewinding of a cassette file, causing the next read or write operation performed on the file to access the first record in the file (not a subfile). After a sequential file has been opened and records written to it, the file cannot be rewound until an EOF mark is written or the file is closed. When the file is a relative record file, file management places a zero in bytes 12-15 (logical relative record number) of the supervisor call block.

The rewind operation is ignored by other devices.

7.2.16 UNLOAD OPERATION (CODE $0F_{16}$). The unload operation is specified by operation code $0F_{16}$. The operation is ignored by all devices and files except the cassette unit. The unload operation for the cassette unit consists of rewinding the cassette tape to the clear area at the beginning of the tape.

7.2.17 UNLOCK OPERATION (CODE $4A_{16}$). The unlock operation, code $4A_{16}$, unlocks a relative record file record that has been locked by a previous read. The record to be unlocked is specified in bytes 12 through 15 of the supervisor call block.

7.2.18 CREATE FILE OPERATION (CODE 90_{16}). The create file operation is specified by placing code 90_{16} in byte 2 of file management supervisor call 00_{16}. To create a file, code utility flag byte 16, logical relative record size bytes 18 and 19, and pathname pointer byte 22. All other bytes in the supervisor call block, except bytes 0 and 2 are ignored.

7.2.19 ASSIGN LUNO TO PATHNAME OPERATION (CODE 91_{16}). The assign LUNO to pathname operation is specified by placing code 91_{16} in byte 2 of the Supervisor Call Block (SCB). The user must code byte 3 of the SCB with the LUNO, and byte 22-23 with the pathname pointer to the file or device. When the user sets the auto-create bit 6 of bytes 16-17 to a 1, the user must also code bits 11-15 of bytes 16-17 and bytes 18, 19 of the SCB. If the diskette file does not already exist it will be created when an open operation is executed using the LUNO in byte 3. Diskette files are not located by file management until they are opened, therefore the user may assign LUNOs to diskette files before the diskette which contains the file is actually loaded. Unlike diskette files, devices are located when the LUNO is assigned. All other bytes of the SCB are ignored.

7.2.20 **DELETE FILE OPERATION (CODE 92_{16}).** The delete file operation is specified by placing code 92_{16} in byte 2 of the Supervisor Call Block (SCB). The file identified by the pathname specified in bytes 22 and 23 is deleted when this operation is executed. Coding of all of the other bytes is ignored. When the file to be deleted does not exist, an error is returned.

7.2.21 **RELEASE LUNO ASSIGNMENT OPERATION (CODE 93_{16}).** The release LUNO assignment operation is specified by placing code 93_{16} in byte 2 of the Supervisor Call Block (SCB). The user must code byte 3 with the LUNO number which is to be released from its previously assigned pathname. All other bytes are ignored.

7.2.22 **COMPRESS FILE OPERATION (CODE 94_{16}).** The compress file operation is specified by placing 94_{16} in byte 2 of the Supervisor Call Block (SCB). The user must also code bytes 22-23 of the SCB with the pathname pointer to the file that is to be compressed. It is beneficial to compress files that have contained a large number of records and then rewritten so they contain fewer records than the original file. The compress operation will return the unused allocation units beyond the end-of-file which are not being used by the current file.

7.2.23 **CHANGE FILE NAME (CODE 95_{16}).** The user must code bytes 22 and 23 with the pointer to the new pathname and code byte 3 with the LUNO which must be previously assigned to the old file pathname which is to be changed. Coding of all other bytes is ignored.

7.2.24 **UNPROTECT FILE OPERATION (CODE 96_{16}).** The user must code bytes 22-23 with the pathname pointer to the file that is to be unprotected. If the file is already opened using an open I/O device-file operation, the file will not be unprotected until the file is closed. All other bytes are ignored.

7.2.25 **WRITE PROTECT FILE OPERATION (CODE 97_{16}).** The user must codes bytes 22-23 of the supervisor call block with the pathname pointer to the file that is to be write protected. If the file is already opened, the file will not be write protected until the file is closed. All other bytes are ignored.

7.2.26 **DELETE PROTECT FILE OPERATION (CODE 98_{16}).** The user must codes bytes 22-23 of the supervisor call block with the pathname pointer to the file that is to be delete protected. If the file is already opened, the file will not be deleted protected until the file is closed. Delete protected files are also write protected. All other bytes are ignored.

7.2.27 **VERIFY PATHNAME SYNTAX (CODE 99_{16}).** The user must code bytes 22-23 of the supervisor call block with the pathname pointer to the pathname that is to be verified. If the syntax is correct, the error status byte will be 0, otherwise it will contain an error code. See error code, Appendix I.

7.2.28 **CODING EXAMPLES USING FILE MANAGEMENT SUPERVISOR CALL 00_{16}.** The following is a coding example for a supervisor call block that will write a record to LUNO 8. LUNO 8 must have been previously assigned to an interactive device. The reply flag is set to 1 indicating that a read will follow the write.

```
SCBO    DATA    0
        BYTE    >0B,8           Write a record to LUNO 8
        DATA    >0040           The flag word of the SCB indicates that this will
                                    be an output with reply.
        DATA    OBUFF           The record will be written from the memory
                                    buffer OBUFF.
        DATA    0
        DATA    80              80 characters will be written from the memory
                                    buffer.
        DATA    RBLK            The reply control block is located at RBLK.
```

An example of a reply block is as follows:

```
RBLK    DATA    RBUFF           Place the reply in the memory buffer RBUFF.
        DATA    40              Input up to 40 characters.
        DATA    0               The number of characters actually inputed will
                                    be returned by the system in this data word.
```

The following example shows how File Utility Supervisor Call 00_{16} assigns LUNO 6 to the diskette file DSC:TEXT/SRC. If the file does not already exist it will be created when an OPEN operation is performed on LUNO 6.

```
SVCBLK EVEN
        BYTE    >00,>00
        BYTE    >91,>06         Assign LUNO 6 to a file.
        BSS     12
        DATA    >1285           Set the auto-create flag to create a noncontiguous
                                    sequential file.
        DATA    0
        DATA    0
        DATA    PATHNM          PATHNM refers to a memory buffer that has the
                                    ASCII representation of the file pathname.
        BSS     12

PATHNM  BYTE    16
        TEXT    'DSC  :TEXT    /SRC'
```

The following example shows an I/O file write operation supervisor call, writing to the logical relative record 56 in the relative record file that LUNO $3C_{16}$ has been previously assigned.

```
SCBF    DATA    0
        BYTE    >0B,>3C         Write to the file that LUNO 3C16 is assigned.
        DATA    0               All the flag bits are turned off.
        DATA    FILOUT          Write it from the memory buffer FILOUT.
        DATA    0               Unused word.
        DATA    20              Write 20 bytes.
        DATA    0
        DATA    56              Write it to record 56 of the relative record file.
```

7.3 SUPERVISOR CALL 15_{16} SUPPORT FOR TASKS DESIGNED TO RUN UNDER TX990, RELEASE 1.0

The TX990 Operating System also supports the file utility supervisor call for tasks that were generated to run under the 1.0 version of the TX990 Operating System.

File utility supervisor call 15_{16} performs the following functions:

- Assigns a LUNO to a device
- Releases a LUNO assignment.

7.3.1 SUPERVISOR CALL 15_{16} SCB FORMAT. The file utility supervisor call block for file utility supervisor call 15_{16} consists of 12 bytes aligned on a word boundary. As shown in figure 7-3, the content of the supervisor call block is as follows:

- 15_{16} (supervisor call code) in byte 0
- A byte reserved for a status code, byte 1
- Utility operation code in byte 2, either of the following:

  01 Assign LUNO to device

  03 Release LUNO assignment

- User flags in byte 3, as follows:

  Zeros in bits 0 through 4

  File Allocation bit – bit 5. Ignored by TX990.

  File Type – bits 6 and 7, set to zero for a device.

- LUNO in byte 4 (FF_{16} is illegal)
- Validation identifier in byte 5. Not applicable to TX990.
- Record length in bytes 6 and 7. Not applicable to TX990.
- Device name in bytes 8 through 11, consisting of up to four characters left-justified with trailing spaces (not required for release function).

The status codes returned by the system in byte 1 at the completion of the function are listed in the error appendix.

7.3.2 CODING EXAMPLE. The example below assigns LUNO 2 to cassette transport drive 2.

```
DATA   >1500        Supervisor call code.
DATA   >0100        Assign opcode.
DATA   >0200        Luno.
DATA   0
TEXT   'CS2 '       Cassette drive 2.
```

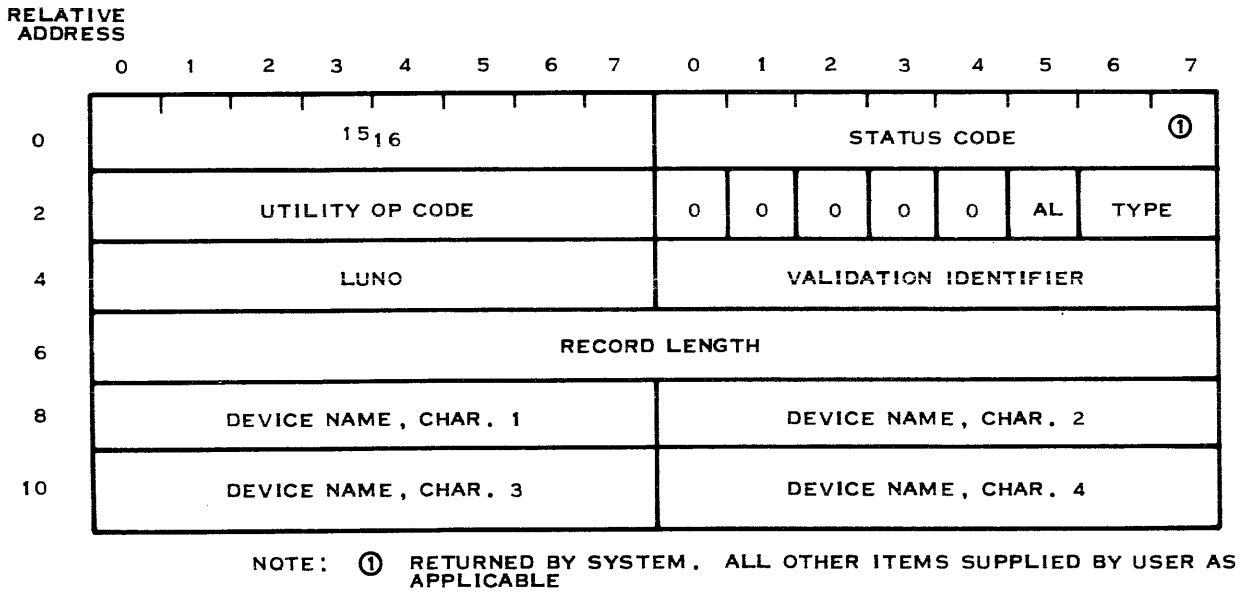


Figure 7-3. File Management Supervisor Call Block for File Management Supervisor Call 15_{16}

7.4 VDT CHARACTER MODE SUPERVISOR CALLS $1A_{16}$, 8_{16}, and 18_{16}.

With these supervisor calls, the user task can position the VDT cursor, write or read fields of data on the VDT screen, and input characters from the VDT keyboard. The following supervisor calls are used for character mode operation:

- VDT Utility Supervisor Call $1A_{16}$
- VDT Character Input from Station Keyboard Supervisor Call 8_{16}
- VDT Conditional Character Input from Station Keyboard Supervisor Call 18_{16}

In the character mode, the user task first executes a VDT utility supervisor call $1A_{16}$, to open the VDT. Subsequent device I/O operations are performed using character input supervisor calls and utility supervisor calls. The VDT is taken out of the character mode by the VDT utility supervisor call that specifies the close device I/O operation.

One or more VDTs may be designated in character mode or record mode, when the system is generated. VDTs designated in record mode may also use character mode I/O operations. Character mode device I/O operations address the VDT by the station number which is assigned to the VDT when the system is generated.

7.4.1 VDT UTILITY SUPERVISOR CALL $1A_{16}$. VDT utility supervisor call $1A_{16}$ allows the following seven VDT I/O operations with a 911 or 913 VDT:

- Opens the VDT, placing it in the character mode
- Positions the cursor
- Tabs the cursor to an unprotected field on the screen
- Reads a field of characters from the screen
- Writes a field of characters to the screen
- Writes a character in a specified number of character positions on the screen
- Closes the VDT, terminating the character mode.

One or more of the above VDT I/O operations may be specified in this supervisor call by setting a bit to 1 in byte 2 of the supervisor call block. When more than one VDT I/O operation is specified, the operations are performed in the sequence indicated in the above list of seven VDT I/O operations.

The open device I/O operation must be performed before any other device I/O operation or supervisor call to the VDT. The user task supplies the station number of the VDT to be opened and requests the open VDT operation. The result is to enable character mode operation of the VDT. When the RC flag is set to 1, the VDT open operation places the maximum values of row and column in bytes 6 and 7 of the supervisor call block, which is normally the buffer address.

The cursor position VDT operation positions the cursor at any character of any line of the screen. The user task supplies the station number of the VDT and the row and column to which the cursor is to be moved.

The tab VDT operation positions the cursor at the leftmost character of an unprotected field. When characters are written on the screen, they are written either as protected or as unprotected characters. When the cursor is at a position that contains a protected character, the tab VDT operation skips protected characters leaving the cursor at the leftmost character of the unprotected field to the right of the original position of the cursor. When the cursor is at a position that contains an unprotected character, the tab VDT operation moves the cursor to the leftmost character of the current unprotected field. The user task supplies the station number of the VDT for the tab VDT operation.

The read characters VDT operation reads a field of characters from the screen at the cursor position. The task may specify a number of characters to be read, or may read all the characters in an unprotected field. This VDT operation also moves the cursor to the position following the last character read. The user task supplies the station number of the VDT, the number of characters to be read, and the address of a buffer in which are placed the characters read from the screen.

The write characters VDT operation writes a field of characters to the screen at the cursor position. The task specifies a maximum number of characters to be written. The operation terminates before writing the maximum number of characters if the operation detects a character having the most significant bit set. In that case, the operation negates the character code, writes the resulting character, and terminates. The user task supplies the station number of the VDT, the maximum number of characters and the address of the buffer.

The write character (propagate) VDT operation writes the first character in the buffer into a field of characters on the screen at the cursor position. The task specifies the number of times the character is to be written. The user task supplies the station number of the VDT, the number of characters to be written, and the address of the buffer that contains the character.

The VDT close operation must be performed to terminate the character mode of the VDT. The VDT may then be opened for record mode operations when the VDT has been designated for both modes. The user task supplies the station number of the VDT to be closed.

The user may specify that the operations of a VDT supervisor call be followed by a beep tone, or that the cursor remain disabled following the operations. All device I/O operations return the row and column of the cursor position at completion.

7.4.1.1 VDT Utility Supervisor Call Block. VDT utility supervisor call block, shown in figure 7-4, consists of the following:

- The code, $1A_{16}$, in byte 0.
- Byte 1 reserved for a completion code (table 4-3).
- Operation flags in byte 2, as follows:

  PR flag – bit 0, set to one to write protected characters. Set to zero to write unprotected characters.

  RC flag – bit 1, set to one to cause an open VDT operation. The maximum values for row and column are placed in bytes 6 and 7 of the supervisor call block.

  PC flag – bit 2, set to one for a position cursor VDT operation.

  T flag – bit 3, set to one for a tab VDT operation.

  R flag – bit 4, set to one for a read VDT operation.

  W flag – bit 5, set to one for a write VDT operation.

  WP flag – bit 6, set to one for a write propagate device I/O operation.

  C flag – bit 7, set to one for a close VDT operation.

- The station number in byte 3.
- Control flags in byte 4, as follows:

  B flag – bit 0, set to one to provide a beep tone following the operations specified.

  CD flag – bit 1, set to one to disable the cursor following the operations specified.

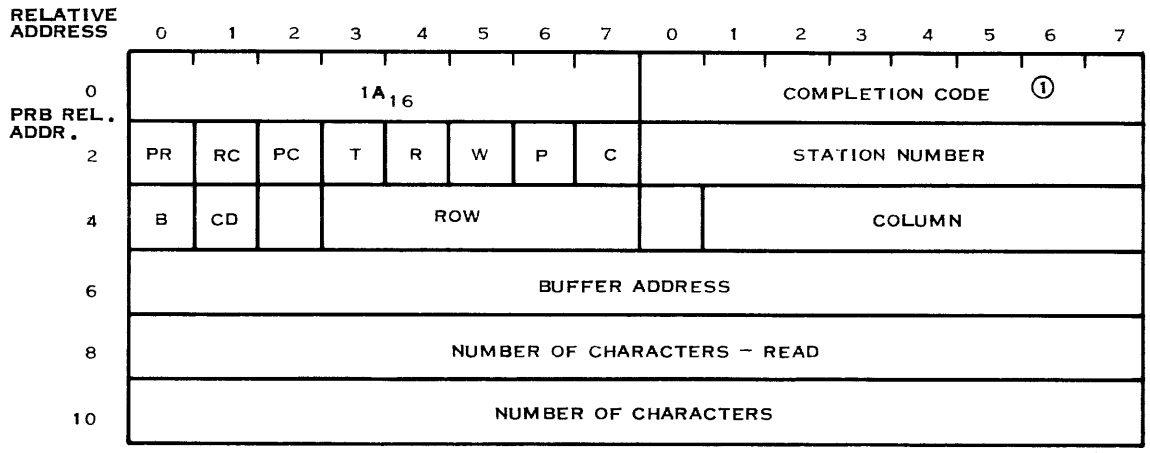
- The row, for a position cursor device I/O operation, in bits 3 through 7 of byte 4. The range of values is 0 through B_{16}. Returned by the system after each call.
- The column for a position cursor device I/O operation, in bits 1 through 7 of byte 5. The range of values is 0 through $4F_{16}$. Returned by the system after each call.

- The address of a buffer in bytes 6 and 7. A read device I/O operation transfers characters into the buffer, a write function transfers characters out of the buffer, and a write propagate device I/O operation copies the first character in the buffer into one or more character positions on the screen. When bit 1 of byte 2 is set to one during an open operation, the maximum values of row and column are placed in bytes 6 and 7.

- The number of characters for a read operation, in bytes 8 and 9. When zero, read the entire unprotected field. Otherwise, read the specified number of characters.

- The maximum number of characters to be written in a write operation, or the number of characters to be written in a write propagate operation, in bytes 10 and 11. In a read operation, the system returns the actual number of characters read in these bytes.

The system returns a completion code in byte 1. The codes are listed in table 7-5.



(A)132858

Figure 7-4. VDT Utility Supervisor Call Block

Table 7-5. VDT Utility Completion Codes

Code (Hexadecimal)	Description
0	Satisfactory completion.
1	Illegal station number.
4	Illegal cursor position specified.
5	Illegal buffer address (Applies to DX10)
6	No unprotected field found in tab function.
7	No unprotected field found in read function of unspecified length (Note 1)
80	VDT currently in record mode or assigned to another task.

NOTE 1

When a read function that has no specified length does not read a protected field, the full screen of characters is read before the operation is terminated in error. Contents of memory locations immediately above the buffer are destroyed when the buffer size is less than the screen size (960 characters for 913 VDT). (1920 characters for 911 VDT).

7.4.1.2 **Coding Example.** The following are examples of coding for supervisor call blocks for VDT utility calls.

```
SCBV    DATA    >1A00,>A402,>4300,WBUFF,0,8
```

Station 2 is already opened, position the cursor at column 0 of row 3, and write eight protected characters from buffer at location WBUFF, leaving the cursor disabled.

```
SCVCT   DATA    >1A00,>5802,0,RBUFF,5,0
```

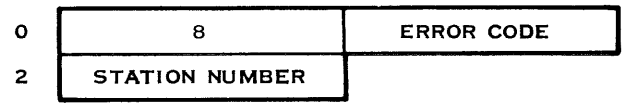
Open station 2, tab cursor, and read five characters into buffer at location RBUFF, with cursor enabled.

```
SCBVC   DATA    >A100,>2902,>8220,RBUFF,4,0
```

Station 2 is already opened, position the cursor at row 2, column 32, read 4 characters into buffer at location RBUFF, close station 2, and sound a beep tone.

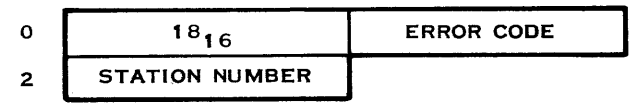
7.4.2 VDT CHARACTER INPUT SUPERVISOR CALL 8_{16}. The character input supervisor call, code 8_{16}, inputs a character from a specified station keyboard. The calling task is suspended until the character is transferred. The system places the character in the most significant byte of the task workspace register 0. The supervisor call block consists of three bytes, and need not be aligned on a word boundary. Byte 0 contains the code, and the system returns a value in byte 1. Byte 2 contains the station number. When the system is unable to locate the station, it returns -1 in byte 1. When the station has not been opened in the character mode, or when power is off at the station, the system returns 80_{16} in byte 1. Otherwise, the system returns zero in that byte.

VDT character input call block:



(A)137501

VDT conditional character input call block:



(A)137502

The following is an example of coding for a supervisor call block for a character input from station keyboard supervisor call:

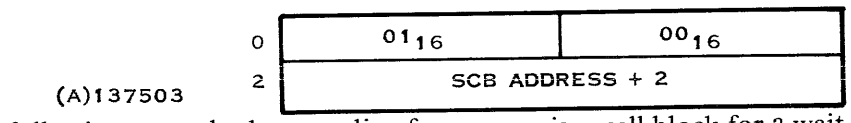
```
SCBC    BYTE    8,0,2          Input a character from station 2 and place the
                               character in the most significant byte of
                               workspace register 0.
```

7.4.3 VDT CONDITIONAL CHARACTER INPUT SUPERVISOR CALL 8_{16}. Conditional character input supervisor call 18_{16} inputs a character from a specified station keyboard. When a character is entered, the function sets the equal bit (bit 2) of the status register to 1 and places the character in the most significant byte of workspace register 0. When a character has not been entered, the function sets the equal bit of the status register to 0, indicating a "not equal" status. In either case, the function returns control to the calling task immediately. The supervisor call block consists of three bytes, and need not be aligned on a word boundary. Byte 0 contains the code, and the system returns a value in byte 1. Byte 2 contains the station number. When the system is unable to locate the station, it returns -1 in byte 1. When the station has not been opened in the character mode, or when power is off at the station, the system returns 80_{16} in byte 1. Otherwise, the system returns zero in that byte.

The following is an example of coding for a supervisor call block for a conditional character input from station keyboard supervisor call:

```
SCBT    BYTE    >18,0,5        Input a character from station 5 and place it on the
                               most significant byte of workspace register 0 if a
                               character has been entered at the keyboard.
```

7.5 WAIT FOR I/O SUPERVISOR CALL 01_{16}.
Wait for I/O supervisor call 01_{16} places the calling task in suspension pending completion of a specified I/O operation. Wait for I/O supervisor call block 01_{16} contains four bytes aligned on a word boundary as shown below. Byte 0 contains the supervisor call code and byte 1 contains a zero. Bytes 2 and 3 contain the address of the second word in the supervisor call block that defines the I/O operation. When the specified I/O operation is not in progress, control is immediately returned to the calling task.



(A)137503

The following example shows coding for a supervisor call block for a wait for I/O call:

```
SWBW    DATA    >100,SCB5+2     Suspend the calling task pending completion of the
                                I/O operation defined in the SCB at location
                                SCB5.
```

7.6 ABORT I/O SUPERVISOR CALL OF_{16} OPERATION.

Abort I/O supervisor call $0F_{16}$ terminates I/O operations on the specified I/O device. The calling task is suspended during execution of the abort I/O supervisor call. If the device is file-oriented, it becomes unassigned. If the device is busy, the system sets the error flag in the supervisor call block for the current operation. No device operation is performed, and the medium remains positioned as the last I/O operation left it. That is, tape in a cassette is not backspaced or rewound, nor are the remaining cards of a deck read. Abort I/O supervisor call block $0F_{16}$ consists of two bytes which need not be aligned on a word boundary as shown below. Byte 0 contains the code and byte 1 contains the LUNO assigned to the device.

(A)137504

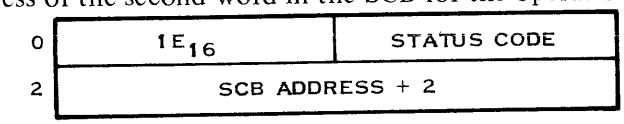
The system returns zero as a status code in byte 1 when the operation completes successfully and -1 (FF_{16}) when the LUNO specified in the SCB has not been defined.

The following example shows the coding for a supervisor call block for an abort I/O call:

```
SCBA    BYTE >F,>11             Abort I/O to the device to which LUNO 11(16) is
                                assigned.
```

7.7 ABORT I/O SUPERVISOR CALL BLOCK $1E_{16}$.

Abort supervisor call $1E_{16}$ terminates an I/O operation defined by a specified Supervisor Call Block (SCB). The abort operation supervisor call block consists of four bytes aligned on a word boundary as shown below. Byte 0 contains the code, and the system returns a status code in byte 1. Bytes 2 and 3 contain the address of the second word in the SCB for the operation to be terminated.



(A)137505

The following is an example of coding for a supervisor call block for an abort supervisor call:

```
SCBA    DATA    >1E00   ABORT I/O Operation     Abort the I/O operation defined in a
        DATA    SCBZ+2  Address                 supervisor call block at location SCBZ.
```

SECTION VIII

DISKETTE OCP SYSTEM UTILITY (SYSUTL) PROGRAM

8.1 INTRODUCTION

Basically, SYSUTL provides the operator with additional keyboard-command-control capability for use with diskette devices and files and, therefore, functions as a diskette-related supplement to the Operator Communication Package (OCP). The SYSUTL commands are listed as follows:

BC – Boot Copy
SF – Set System File
CF – Create File
DF – Delete File
CM – Compress File
CN – Change File Name
CP – Change Protection
DO – Define Output
MD – Map Diskette
MF – Map File
DD – Diskette Dump
DL – Diskette Load
FD – File Dump
FL – File Load
ID – Initialize Date and Time
TI – Print Time and Date
TE – Terminate
CV – Change Volume Name

In general, SYSUTL is a module of TX990 and a diskette-resident extension of the OCP module (described in Section III of this manual). SYSUTL may be executed under OCP or under the Terminal Executive Development System (TXDS) using console control in a manner similar to OCP. The following paragraphs describe the loading procedure, LUNOs, syntax, and the individual commands. The last paragraph in this section covers the SYSUTL error messages.

8.2 LOADING SYSUTL

SYSUTL can be loaded using the OCP module or in conjunction with the Control Program. When using the OCP module, perform the procedure itemized in paragraph 8.2.1. When using the Control Program perform the procedure itemized in paragraph 8.2.2.

8.2.1 LOADING SYSUTL USING OCP.

Load the program as follows:

1. After loading the OCP in accordance with the procedure itemized in the section entitled Operator Communication Package, observe the printout or display on the system console of the period (.) prompt.

2. Place the object module for SYSUTL in either the cassette or floppy diskette drive and ready the device.

3. Using OCP's LP (Load Program) command, load the SYSUTL object module from the input media into memory as follows:

.LP,DSC2:SYSUTL/SYS	Load from floppy diskette DSC2 the object module file SYSUTL/SYS.

An alternate method for loading the SYSUTL object module from cassette follows:

.LP,CS1.	Load SYSUTL object module from cassette unit 1.

4. Enter OCP'S EX (Execute) command to execute SYSUTL and terminate OCP as follows:

   .EX,10.TE. Executes SYSUTL and terminates OCP

5. Observe the following printout or display on the system console:

   TX990 SYSTEMS UTILITY 937544**

6. After SYSUTL has started execution, the SYSUTL module causes the OP: prompt to be printed or displayed on the system console as follows:

   TX990 SYSTEM UTILITY 937544**
   OP:

7. At this step in the procedure, one or more SYSUTL commands, with a limit of up to 76 characters on one line, may be entered on the system console. An example follows:

   OP: MD,DSC2.SF,DSC2:SYSFIL/SYS

NOTE

All of the commands must be entered on the same OP: line and not carried to another line.

8. After SYSUTL executes the commands entered in the preceding step, observe that the OP: prompt is again printed out or displayed on the system console.

9. At this step in the procedure, the operator has the choice of entering another SYSUTL command or terminating SYSUTL by entering a terminate response to the prompt as follows:

   OP: TE.

NOTE

If an error occurs during execution of a command, the appropriate error message is issued.

10. If TXDS is in the system, observe that SYSUTL has terminated when the following printout or display is produced on the system console:

    TXDS 936215 ** (Date and Time)

    PROGRAM:

8.2.2 LOADING SYSUTL USING THE TX990 OPERATING SYSTEM AND THE TXDS CONTROL PROGRAM. Load the program as follows:

1. After loading the TX990 Operating System in accordance with the loading procedure in the section entitled Loading the TX990 Operating System, bid the TXDS Control Program. This can be accomplished by use of the OCP module or without use of the OCP module. When the OCP module is used, enter EX,16.TE. in response to OCP's period (.) prompt to bid the TXDS Control Program and proceed to the next step. When the OCP module is not present, enter an exclamation point (!) to bid the TXDS Control Program and proceed to the next step.

2. Observe the following printout or display on the system console:

   TXDS 936215 ** 359/77 1:00

   PROGRAM:

3. Enter the pathname of the SYSUTL object module in response to the PROGRAM: prompt as follows:

   PROGRAM: :SYSUTL/SYS

4. Depress the carriage return key and observe that the INPUT: prompt is printed out or displayed on the system console.

NOTE

The asterisk (*) feature can be used in lieu of the carriage return/ NEW LINE entry to by pass the remaining prompts (Input, Output, Options).

5. Make a null entry by depressing the carriage return and observe that the OUTPUT: prompt is printed out or displayed on the system console.

6. Make a null entry by depressing the carriage return and observe that the OPTIONS: prompt is printed out or displayed on the system console.

7. In response to the OPTIONS: prompt, only one SYSUTL command can be entered on the system console (with a limit of up to 30 characters on the same line). For example:

   OPTIONS: MD,DSC2

NOTE

1. When multiple SYSUTL commands are to be entered, the response to the OPTIONS: prompt must be a carriage return (i.e., a null entry). This will cause SYSUTL to print or display the OP: prompt on the system console to which prompt the operator responds with one or more SYSUTL commands (with a limit of up to 76 characters on one line).

2. A SYSUTL command may not be continued on the next line.

3. When multiple SYSUTL commands are entered, SYSUTL processes each individual command, one at a time, until it has no more commands to process, at which time the operator can enter a TE (Terminate) SYSUTL command. This, in turn, causes the TXDS Control Program to be rebid.

8. After the SYSUTL command entry is made in response to the OPTIONS: prompt, observe that the TXDS Control Program is rebid and the following printout or display is presented on the system console:

   TXDS 936215 ** (Date and Time)

   PROGRAM:

8.3 LUNOs

SYSUTL uses LUNO 0, LUNO 6, and LUNO 7. LUNO 0 is assigned to the system console. All communications between operator and task, including error messages, are performed through LUNO 0. LUNO 6 is assigned by SYSUTL to the diskette being accessed by all diskette-related commands. LUNO 7 is internally assigned to the default print device upon loading SYSUTL and is used as the SYSUTL output LUNO for any printer output. The user has the option to redirect any printed output, excluding error messages, to a device other than the LOG by executing the Define Output (DO) command (described in one of the paragraphs below).

8.4 SYSUTL COMMAND FORMAT AND SYNTAX

Each command consists of a command word, optionally followed by one or more operands. SYSUTL recognized a command by the first two letters of the command word, the command key. These letters may be followed by additional letters or by blanks. One or more blanks, or a comma, may separate a command key and its operands. However, embedded blanks are not allowed within the command word or an operand.

If Systems Utility is executed using OCP, more than one command may be entered in a single line, which may contain up to 72 characters and must be terminated by a carriage return (NEW LINE on 913 VDT). When more than one command is entered on a line, each command must be terminated with a period, except the last command on the line, then it may be omitted. When an error is detected in a command, any subsequent command on that line is ignored.

SYSUTL syntax also entails the following items and/or rules:

- Angle brackets < > enclose items required to be supplied by the user
- Brackets [ ] enclose optional items
- Braces { } enclose alternative items, one of which must be entered
- An ellipsis (. . . .) indicates that the preceding items may be repeated
- Items shown in capital letters must be entered as shown
- Pathname is described in 1.4.1.

8.5 SYSUTL COMMANDS

SYSUTL provides commands to initialize the time and date and to display the time and date. SYSUTL also provides a command to define a listing device for the display commands. All the remaining commands deal with functions that involve the diskette. SYSUTL has a command to invoke every file utility operation: create a file; delete a file; compress a file; change a file name; and change protection on a file. SYSUTL also supports maintenance functions for the diskette: map the diskette, which displays the file names along with various information of all the files on the diskette; map file, which displays information for a single file; disc dump, which displays absolute locations on the diskette; disc load, which writes to absolute locations on the diskette; file dump, which displays physical sectors within a file; and file load, which writes to physical sectors within a file.

SYSUTL provides two commands for maintenance of a system diskette. The first command is Boot Copy (BC) which copies the TXBOOT program to diskette. The second command is Set System File (SF), which allows the user to define any object file as the file that will be loaded as the Operating System. The system file pathname must be defined before the Operating System loading procedure is performed. (Refer to the loading procedure in Section II of this manual for additional information.)

If a null entry is made wherever a device is to be specified, the default-substitute will be the system diskette which is specified during system generation. There will be no other defaults.

All numeric output is in decimal except the output from the Diskette Dump (DD) and File Dump (FD) commands. These dumps are in hexadecimal.

8.5.1 **BOOT COPY (BC).** The Boot Copy (BC) command causes a copy of the TXBOOT program to be written on the specified diskette. A diskette ROM loader cannot be used to load the Operating System or a stand-alone program from diskette until the TXBOOT program is written to the diskette that will be used during loading. The syntax of the command is presented as follows:

BC, {<device>. / <volume>.}

The operand must be a one-to-four-character diskette or volume name. There is no default-substitute for the device operand. Upon executing the BC command, the TXBOOT program is copied to physical tract 0 of the specified diskette. The following is an example of the Boot Copy (BC) command:

OP:BC,DSC2.	The TXBOOT program is copied to physical track 0 of DSC2.

8.5.2 **SET SYSTEM FILE (SF).** The Set System File (SF) command defines a diskette file to be loaded as the Operating System or as a stand-alone program. The syntax is as follows:

SF,<pathname>.

The file designated by the specified pathname is the file that contains the linked object code that is to be loaded and executed as the Operating System or as a stand-alone program. Refer to the diskette/cassette ROM loader description for further details on loading. An example of the Set System File (SF) command follows:

OP:SF,DSC:TX990/SYS.	The file TX990/SYS on DSC is to be used as the Operating System file when the diskette is booted.

8.5.3 **CREATE FILE (CF).** The Create File (CF) command creates a sequential or relative record file. The syntax for the command is as follows:

CF,<pathname>,[,<RR>,<record length>].

The pathname operand is the name assigned to the file upon its creation. The newly created file is a noncontiguous sequential file unless the optional operand 'RR' follows the pathname. If 'RR' is specified, the file is created as a noncontiguous relative record file with the specified record length. For relative record files, the recrod length must be given. Record length is assumed to be decimal uniess preceded by '>'. All files are created unprotected and have a protection code of 'U'.

The following are examples of the CF command:

OP:CF,DSC2:FILE/SEQ.	Create sequential file named 'FILE/SEQ' on DSC2.
OP:CF,VOL1:RELREC,RR,>80.	Create relative record file names 'RELREC' with record length of 80_{16} on the diskette with the volume name VOL1.

8.5.4 **DELETE FILE (DF).** The Delete File (DF) command deletes the specified file from the diskette. The syntax for the command is as follows:

DF,<pathname>.

The following question is issued:

ARE YOU SURE???

Only one character is accepted. If any response is entered other than a 'Y' for 'yes', the delete file command is ignored. Upon entering a 'Y', the specified file is deleted from the diskette. An example of the DF command follows:

OP:DF,DSC:TEMPFIL

ARE YOU SURE??? Y	Upon entering 'Y', the file 'TEMPFIL' is deleted from DSC.

8.5.5 **COMPRESS FILE (CM).** The Compress File (CM) command returns to the Operating System all allocation units (AUs) beyond the EOF for the specified pathname.

CM,<pathname>.

The pathname operand must be a valid pathname as defined in the section in this manual entitled File Management Executive Supervisor Calls. The Compress File command compresses the designated file to its minimum size. The following is an example of the CM command:

OP: CM,:OCPFIL/SRC.	Compresses file 'OCPFIL/SRC' on system default diskette.

8.5.6 **CHANGE FILE NAME (CN).** The Change File Name (CN) command changes the file name and extension of an existing file to the new pathname as specified. The syntax of the command is as follows:

CN,<old pathname>,<new pathname>.

Both the old pathname and new pathname operands must be valid pathnames as defined in section I of this manual. The file with the specified old pathname retains all its directory information such as protection code and file type. Only the name of the file on the diskette is changed. The following is an example of the CN command:

OP: CN,DSC2:OCPFL1,DSC2:OCPFIL/SYS. — Changes the name of the file that is currently listed in the file directory on DSC2 as OCPFL1 to the new file name OCPFIL/SYS.

8.5.7 **CHANGE PROTECTION (CP).** The Change Protection (CP) command changes the file protection code of the specified file to the new protection code. The syntax of the command is as follows:

CP,<pathname>, { U / D / W }

The pathname operand must be a valid pathname. The existing protection code of the specified file is changed to the newly defined protection code. The second operand is a one-character abbreviation of the possible protection code as follows:

U	–	Unprotect File	(File is Unprotected)
D	–	Delete Protect File	(Protects File from Being Deleted)
W	–	Write Protect File	(Protects File from Being Written On or Deleted)

If the specified file is already open, the new protection will not be in effect until the file is closed.

The following is an example of the CP command:

OP: CP,DSC:LIST80/SYS,W. — Protects the file 'LIST80/SYS' on DSC from being written over or deleted.

8.5.8 **DEFINE OUTPUT (DO).** The Define Output (DO) command directs the displays of several SYSUTL commands to the specified device. The syntax of the command is as follows:

DO,<device>.

All displays are directed to the default print device until changed. The DO command allows the printed output of the following commands to be directed to the specified device:

MD – Map Disc

MF – Map File

DD – Disc Dump

FD – File Dump

TI – Print Time and Date

ID – Initialize Date and Time.

The device operand is a one- to four-character device name.

If SYSUTL is terminated and then rebid without reloading, the use of the last output device assigned will be resumed. The following is an example of the DO command:

OP: DO,LP.	Redirects all displays (excluding error messages) to the line printer until the DO command is reexecuted.
OP: DO.	Displays are resumed to the default print device, which is the system console for standard TI supplied Operating Systems.

8.5.9 MAP DISKETTE (MD). The Map Diskette (MD) command maps the specified diskette by file name, by extension, or maps every file on the diskette to the display device. The syntax for the command is as follows:

MD [{<device>}] [{,<extension>}]
   [{<volume>}] [{<file name>}]

If no device is specified, the operating system diskette is used. The device operand is a one- to four-character device name that is to be mapped. The extension operand consists of a slash immediately followed by a one- to three-character extension name which will be used to map the diskette. To map the diskette according to a specified file name, a colon must be entered and immediately followed by a one- to seven-character file name.

The Map Disc command outputs the Diskette Identification field, the total number of allocation units on the specified diskette, and the number of available and bad Allocation Units (AUs) on the diskette. MD also lists the name of all files on the diskette, their file type, file protection code, and the total number of AUs allocated to each file. The command output is terminated with the printing of the time and date.

The following are examples of legal MD commands:

MD,DSC2:FILE/.	Map all files with the name FILE and any extension on DSC2.
MD,VOL2:FILE/.	Map all files with the name FILE and any extension on the diskette with volume name VOL2.
MD,VOL/OBJ.	Map all files with the extension OBJ on volume VOL.
MD,/SRC.	Map all files on the default disc with the extension SRC.
MD,VOL3.	Map all files on volume VOL3.
MD,:TASK1/.	Map all files on the default disc with the name TASK1 and any extension.

It is not legal to give both a file name and an extension in a Map Diskette command.

The output of the MD command may be directed to any desired device using the Define Output command. Output of the MD command is sorted alphabetically by file name and extension. The following are examples of the Map Disc command:

OP: MD. Maps all of the default diskette.

```
DISC I. D. : TX990 1/27/77

DSC        ALLOC. UNITS(TOTAL): 333     FREE:    0    BAD:    0

   FILE          TYPE   PT   ALLOC. UNITS

DSKDMT/SYS         S     U         18
IBMUTL/OBJ         S     U         20
SCRATCH/SRC        S     U          8
SCRTCH             S     U        102
SOURCE/SRC         S     U          8
SYS012             S     U          8
SYS0127/SYS        S     U        102
TEMP/SRC           S     U          8
TEMP/TMP           S     U          7
TXEDIT/SYS         S     U         22
TXTST1/SRC         S     U          8
TXTST2/OBJ         S     U          8
TXTST3/OBJ         S     U          8

10:45:35       FEB 3, 1977
```

OP: MD,/SYS. Maps all files on default diskette with the extension 'SYS'.

```
DISC I. D. :  TX990 1/27/77

DSC        ALLOC.  UNITS(TOTAL). 333     FREE:  0    BAD:    0

   FILE          TYPE   PT   ALLOC. UNITS

DSKDMT/SYS         S     U         18
SYS0127/SYS        S     U        102
TXEDIT/SYS         S     U         22

10:46:08       FEB  3, 1977
```

8.5.10 MAP FILE (MF). The Map File (MF) command outputs pertinent information of the specified file to the defined output device. The syntax for the command is as follows:

MF,<pathname>.

The pathname operand specifies the name of the file that is to be mapped and the device on which it is located. This command displays the type of file, whether sequential or relative record, the file-protection code, the record length if it is a relative record file, and the number of allocation units (AU) contained in the file. The MF command also outputs the starting allocation unit number and the number of units contained in each block allocated to the specified file. The command output is terminated with the printing of the time and date. The following are examples of the Map File (MF) command for a sequential record file.

```
OP: MF,DSC:DSKDMT/SYS.        Map file 'DSKDMT/SYS' on diskette DSC.

NAME:  DSKDMT/SYS     TYPE:S    PT:U     REC LENGTH:          # A. U. :   18

          START ALLOC.  UNIT              # OF UNITS

                200                           16
                240                            2

10:46:52     FEB 3, 1977
```

8.5.11 DISKETTE DUMP (DD). The diskette Dump (DD) command causes SYSUTL to print the contents of the specified absolute diskette locations on the defined display device. The syntax for the command is as follows:

```
DD[, <device>],<alloc unit>,<sector>[,<start byte>[,<end byte>]].
DD,T[,<device>],<track>,<sector>[,<start byte>[,<end byte>]].
```

The Diskette Dump command may be executed in either of two modes: by allocation unit or physical track. If the diskette is to be accessed by track, a "T" must be entered immediately after the command key 'DD'. The optional device operand is a one- to four-character name. If the device operand is omitted, the default system diskette as specified during system generation is used. The allocation unit operand is a one- to three-digit number that specifies the desired allocation unit. If the dump is to be performed by physical diskette track, the track operand is a one- to two-digit number that specifies the desired track. The sector operand is a one- to two-digit number that designates a sector on the allocation unit or track. The next two operands are one- to four-digit numbers that specify the starting and ending byte addresses within the sector to be printed. When only the starting byte is entered, the contents of the word that contains the specified byte as well as the remainder of the sector, are printed. When both the starting and ending byte addresses are omitted, the contents of the entire sector are printed.

All numeric operands (allocation unit, track, sector, starting and ending byte) are assumed to be decimal unless otherwise preceded by a '>' for hexadecimal. The following is an example of the Diskette Dump (DD) command and the resulting printout:

OP: DD,12,0,>10,>30 — Dumps sector 0 of allocation unit 12, byte 10_{16} through 30_{16} on default diskette.

```
0010 3641 3043 3043 3041 4231 3030 3837 4631   6A 0C 0C 0A B1 00 87 F1
0020 3445 4620 5458 3939 3030 3535 FF42 3036   4E F  TX 99 00 55 .B 06
0030 4130                                      A0

10:47:41   FEB 3, 1977
```

The contents of sixteen bytes are printed per line following the address of the first byte printed on the line. The contents of each pair of bytes are shown as four hexadecimal digits. At the right end of the line, the contents are printed as ASCII characters. The bytes that contain values that correspond to printable ASCII characters are translated and printed as ASCII characters. The nonprinting ASCII characters are printed as periods. The display is terminated with the printing of the time and date.

8.5.12 DISKETTE LOAD (DL). The Diskette Load (DL) command places the specified data on a diskette at a specified address. The Diskette Load command may be performed by allocation units or by physical track. The syntax for the command is as follows:

DL [,<device>],<allocation unit>,<sector>,<starting byte>,<data>[,<data>...]
DL,T[,<device>],<track>,<sector>,<starting byte>,<data>[,<data>....]

If the diskette is to be accessed by track, a 'T' must be entered immediately after the command key 'DL'. The optional device operand is a one- to four-character device name. If the device operand is omitted, the default-substituted system diskette as specified at SYSGEN time is used. The allocation-unit operand is a one- to three-digit number which specifies the allocation unit to be accessed. A number consisting of one to two digits is entered for the track operand if the diskette is to be accessed by physical track. The sector is a one- to two-digit number that specifies a sector on the allocation unit or track that is to be loaded. The starting byte is a one- to four-digit byte address into which the first data value is placed. If the byte address is odd, the previous even-byte address will be used. When additional words of data are entered, they are placed in successive addresses.

All numeric operands (allocation unit, track, sector, starting byte, data) are assumed to be decimal unless otherwise preceded by a '>' for hexadecimal. The following is an example of a Diskette Load (DL) command:

OP: DL,DSC2,1,3,>20,>5448,>4453,32,57.

Place the ASCII codes for the characters T, X, D, and S in bytes 20_{16} through 23_{16} and decimal data in bytes 24_{16} through 27_{16} of sector 3 of allocation unit 1 on DSC2.

CAUTION

The DL command has the capability of modifying any data on the system diskette, including the Operating System file and Operating System file structures. Verify the address by performing a DD command that specifies the same allocation unit or track, and sector before entering a DL command. When the contents do not agree with the contents of the bytes to be altered, determine the correct AU track and sector before proceeding.

8.5.13 FILE DUMP (FD). The File Dump (FD) command causes SYSUTL to print the contents of a specified physical sector of a specified diskette file. The syntax of the command is as follows:

FD,<pathname>,<sector>[,<starting byte>[,<ending byte>]].

The pathname operand specifies the name of the file and the diskette unit on which it is located that is to be dumped. The sector operand is the number of the sector that is to be accessed relative to the beginning of the file. The next two operands, the starting and ending byte, are one- to four-digit addresses within the sector to be printed. When only the starting byte is entered, the contents of the word that contains the specified byte are printed. When the byte addresses are omitted, the contents of the entire sector are printed. The contents are printed in hexadecimal representation and in ASCII character representation. A nonprinting character is represented by a period. The display is terminated with the printing of the time and date.

All numeric operands (sector, starting byte, ending byte) are assumed to be decimal unless otherwise preceded by '>' for hexadecimal. The following is an example of a File Dump (FD) command and the resulting output:

OP: FD,DSC2:SYSUTL/SYS,1,,>36. — Dumps sector 1, bytes 0 through 36_{16} of file SYSUTL/SYS on DSC2.

8.5.14 FILE LOAD (FL). The File Load (FL) command places the specified data in specified bytes of a diskette file record. The syntax for the command is as follows:

FL,<pathname>,<sector>,<starting byte>,<data>[,<data>....]

The pathname operand specifies the name of the file and the diskette unit on which it is located that is to be accessed. The sector operand is a one- to two-digit number presenting the sector to be loaded relative to the beginning of the file. The starting byte is a one- to four-digit address into which the first data value is placed. When an odd number is entered for the starting byte, the value is placed in the even address immediately preceding the odd value. When additional words of data are entered, they are placed in successive addresses.

All numeric operands (sector, starting byte, data) are assumed to be decimal unless otherwise preceded by a '>' for hexadecimal. The following is an example of an FL command:

OP: FL,DSC:TEMPFIL,2,0,>444D,>4879. — Places $444D_{16}$ and 4879_{16} into bytes 0 through 3 of the second sector in TEMPFIL on DSC.

8.5.15 INITIALIZE DATE AND TIME (ID). The Initialize Date and Time (ID) command initializes the date and time values for the system. The syntax for the command is as follows:

ID,<year>,<month>,<day>,<hour>,<minute>.

The year operand is the four-digit decimal number of the years 1976 through 1999, and the month operand is the decimal number of the month 1 through 12. The day operand is a one- or two-digit decimal number, 1 through 31, and the hour operand is a one- or two-digit decimal number, 0 through 23. The minute is the decimal number of the minute 0 through 59. The second is set to zero when the command is entered. An example of an ID command follows:

OP: ID,1977,2,12,17,29 — Initialize the time and date to 5:29 PM, February 12, 1977.

17:29:00 FEB 12, 1977

8.5.16 PRINT TIME AND DATE (TI). The Print Time and Date (TI) command causes SYSUTL to print the time and date. The syntax for the command is as follows:

TI.

The command requires no operands. The following example shows a TI command and the resultant output.

OP: TI.

9:29:12 FEB 13, 1977

When the date and time have not been initialized, no date is printed and the time printed is the elapsed time from the most recent loading of the Operating System or from system restart.

8.5.17 **TERMINATE SYSUTL (TE).** The Terminate SYSUTL (TE) command terminates execution of SYSUTL and releases all related LUNOs. If the TXDS Control Program is linked with the TX990 Operating System, it will be rebid. The syntax for the command is as follows:

TE.

The command requires no operands.

8.5.18 **CHANGE VOLUME NAME (CV).** The change volume (CV) command causes SYSUTL to change the volume name of a diskette to a new volume name. The syntax for the command is as follows:

CV, [<device> / <volume>] , <volume>.

The first parameter defines the old volume name or device. The second parameter defines the new volume name. It is wise to avoid volume names that are also device names, because the TX990 operating system will not be able to find the correct volume.

When a volume name for a disc has not been defined, it may be defined by using the device name, as follows:

CV,DSC2,VOL2.

8.6 SYSUTL ERROR MESSAGES

The error messages capable of being issued by SYSUTL are listed in table 8-1. When more than one SYSUTL command is entered on a line and an error occurs, the command in error (or that caused the error) and all subsequent commands in the statement must be entered again. Unless otherwise noted, when an error occurs, the command is ignored and SYSUTL prompts for input.

Table 8-1. SYSUTL Error Messages

Message	Meaning	Recovery
INVALID COMMAND	The command word is not valid.	Check the command word, and reenter correctly.
I/O ERROR, IGNORED	An I/O error was detected during reading of the command.	Input from the system console was not received correctly. Check the device, and reenter the entire line.
OUTPUT ERROR	An error was detected during output and execution of the command has been terminated.	This message is printed when the user has entered an ESC character to terminate output, or if LUNO 1 has not been assigned. If neither of these applies, check device for errors.
UNDEFINED ERROR	TX990 returned an error code to SYSUTL that is not recognized by SYSUTL.	This is a system error. Make another attempt to enter the command. If error recurs, reinitialize the system.
OPERAND ERROR(S)	One or more operands are invalid.	When a required operand has been omitted, enter the complete command. When a numeric operand is required, but a nonnumeric operand was entered, enter the command with the correct type of operand.
BAD DEVICE	Invalid device name.	Enter the command with one of the device names assigned when the system was generated.
DEVICE OFF LINE	Specified device not on line.	Check diskette device. If power off or diskette not installed in drive, correct and reissue command.
INVALID DISC ADDRESS	Invalid track, AU, or sector specified.	Check command operands and reenter correctly.
BAD FILE NAME	Specified pathname has a syntax error.	Check pathname input and reenter correctly.
INVALID INPUT PARAMETER	Bad parameter in supervisor call block.	Reload and execute SYSUTL.
DUPLICATE FILE NAME	File already exists with specified filename.	Check file name and reenter correctly.
FILE NAME UNDEFINED	Specified file does not exist.	Check file name and reenter correctly.
DISKETTE FULL	No available allocation unit or diskette.	Delete any unnecessary files and/or compress files. Printer command.

Table 8-1. SYSUTL Error Messages (Continued)

Message	Meaning	Recovery
FILE DELETE PROTECTED — UNABLE TO DELETE	Delete File command. Attempted to delete file that is write-protected or delete-protected.	If you wish to delete the specified file, it must first be unprotected.
INVALID FILE PROTECTION SPECIFIED	Change File protection command. Protection specified is not valid character: 'U', 'D', or 'W'.	Reenter file-protection code.
UNABLE TO READ DISC	Error occurred while reading diskette.	Reenter command.
UNABLE TO WRITE DISC	Error occurred while writing to diskette.	Reenter command.
CAN'T OPEN OUTPUT LUNO	SYSUTL output LUNO is already in use by another task.	If a record mode I/O-operation to the LUNO is in progress, enter the command again after the operation is completed. If the LUNO is assigned to a file-oriented device that has been opened, enter the command again after the LUNO has been closed.
CAN'T LOAD PAST END OF SECTOR	Diskette Load or File Load command. Specified sector not in range of file.	Check desired sector number and reenter command correctly.
UNABLE TO GET REQUIRED MEMORY	Memory required to execute specified command is not available.	
UNABLE TO GET COMMON	Error occurred in trying to get COMMON.	Verify that TX990 has COMMON. If not, the system must be regenerated to include COMMON.
FILE DIRECTORY EMPTY	Map Diskette command. Attempted to map a diskette which has no files on it.	Check device name and reenter correctly.
INPUT PARAMETER OUT OF RANGE OF SECTOR.	Byte address specified not in range of sector.	Check input and reenter command correctly.

SECTION IX

SYSTEM GENERATION

9.1 INTRODUCTION

The GENTX utility program provides the user with the capability of generating an operating system customized to the user's specific hardware and software configuration. Basically, the customized operating system is generated as follows:

- First, GENTX is used to produce source code for two source modules: TXDATA and TASKDF. The TXDATA source module is used for providing the data needed by the operating system to enable its communicating with (i.e., controlling) each device in the user's total hardware configuration. (Whenever another device is added to the user's hardware configuration, it is necessary to add data to the TXDATA source module.) The TASKDF source module is used for providng the data needed by the operating system to enable execution of any of the various TI-supplied modules/tasks (e.g., File Management, Operator Communications Package, Start Task, Diagnostic Task, etc.) or of any user-supplied tasks/programs (including user-utilities).

- Second, an assembler is used to assemble the TXDATA and TASKDF source modules into object modules.

- Third, a linker is used to link TXDATA and TASKDF object modules to other selected object modules such as a user task/program, the File Management utility program, the Operator Communications Package, Start Task, Diagnostic Task, etc. This linked object module is the resultant customized Operating System.

The following paragraphs describe these steps in detail.

9.2 PREPARATION FOR GENERATING A TX990 OPERATING SYSTEM

In preparation for generating a customized TX990 operating system, the user must select the peripheral devices to be included in his system and assign Communications Register Unit (CRU) addresses and interrupt levels for each of those devices which can be obtained from the chart that is pasted on top of the Model 990/4 or Model 990/10 computer chassis.

The user must write and assemble a Device Service Routine (DSR) for each type of device not supported by any of the TI-supplied TX990 modules. The DSR must process interrupts generated by the device and I/O calls to the device. The devices supported by TI-supplied modules are listed in Section I. The user must also write and assembly and user-supplied routines (e.g. supervisor calls, extended operations) and include the object code when linking the system. Appendix C contains detailed information about writing such routines.

9.3 DEFINING THE NEW SYSTEM

Definition of a new operating system is accomplished by executing the GENTX utility program. GENTX prompts the user to define various parameters, as well as devices connected in the system. After all parameters have been defined GENTX builds two source modules, TXDATA and TASKDF, which form part of the operating system kernel.

9.3.1 LUNOS USED BY GENTX. GENTX uses the system console for user interactions and LUNO 10_{16} for file output.

9.3.2 LOADING AND EXECUTING SYSTEM GENERATION (GENTX) UTILITY PROGRAM. GENTX may be loaded and executed using a TX990 or DX10 Operating System. The following paragraphs describe how to load and execute GENTX using a TX990 Operating System, or TXDS, or DX10 release 2.2 system.

9.3.2.1 Loading and Executing Using TXDS. When executing GENTX using TXDS Control Program in a system without OCP, place the GENTX object module in the appropriate input device and perform the following steps:

1. Enter an exclamation point (!).

2. Enter the GENTX pathname to the "PROGRAM:" prompt as follows:

PROGRAM:	:GENTX/SYS*	If the object program for GENTX is in diskette file, :GENTX/SYS.
PROGRAM:	CS1*	If on Cassette 1.
PROGRAM:	CR*	If on Card Reader.

After GENTX starts executing it will display the following title message:

```
TX990 SYSTEM GENERATION      945673    *B
```

9.3.2.2 Loading and Executing GENTX with TX990 Using OCP. When executing GENTX on a TX990 system using OCP, place the GENTX object module in the appropriate input device and perform the following steps:

1. Enter an exclamation point (!) at the keyboard of the system console to activate OCP. OCP responds by printing a period (.) to request a command.

2. Enter an OCP command to load GENTX in the dynamic task area. The following are examples of commands to load GENTX:

.LP,CS1,3.	Load task GENTX at priority level 3 when the object module is on Cassette Unit 1.
.LP,CR,3.	Load task GENTX at priority level 3 when the object module is in the Card Reader.
.LP,:GENTX/SYS.	Load task GENTX at priority level 3 when the object module is in diskette file.

3. Enter the following OCP command to execute GENTX and terminate OCP:

.EX,10.TE.	Execute the task in the dynamic task area, GENTX and terminate OCP.

After GENTX starts executing it will display the following message:

TX990 SYSTEM GENERATION 945673 *B

9.3.2.3 Loading and Executing GENTX with DX10 Release 3.0. For details on the DX10 release 3.0 process, refer to Appendix K.

9.3.3 DEFINITION PHASE. GENTX begins execution in the definition phase, in which GENTX requests the user to enter system parameters and constructs an internal table of data from which GENTX later constructs the source statements for TXDATA and TASKDF. GENTX prints the request on the system console, and the user enters the parameters on the keyboard of the same device. In the following paragraphs, numeric parameters are specified as hexadecimal or decimal numbers according to the type of number that normally would be used. Either hexadecimal or decimal values may be entered for any numeric parameter. A hexadecimal number is preceded by a greater-than (>) character. If at anytime the user wishes to terminate GENTX, enter an asterisk. Table 9-1 shows all of the prompts which may be displayed by GENTX, and gives information concerning correct replies. The following paragraphs describe each prompt.

9.3.3.1 General System Definitions. The first prompt displayed by GENTX is as follows:

MEMORY AVAILABLE–

The user enters the amount of memory (bytes) available to GENTX, to be used for table storage. No default when only a carriage return is entered. The total memory available may be calculated by subtracting the size of GENTX ($2B65_{16}$ bytes) from the total memory available. The suggested memory allocation is 2000_{10} bytes.

The next six prompts are requesting timing information that will be used by the task scheduler within the new TX990 Operating System being generated.

LINE FREQ.–

The user enters the power-line frequency, a decimal value followed by a carriage return. The valid parameters are 50 for 50-Hz power-line frequency or 60 for 60-Hz power-line frequency. When a carriage return only is entered, GENTX uses the default value, 60.

The next request is as follows:

TIME SLICE–

The user enters the number of real-time-clock cycles per maximum time period, followed by a carriage return. The maximum time period is used for a time slice by the task scheduler. When only a carriage return is entered, GENTX uses the default value, 6, corresponding to a 50-ms time slice when the power-line frequency is 60-Hz, or a 40-ms time slice when the power-line frequency is 50-Hz (see table 9-2).

The next four requests ask for the maximum number of consecutive time slices for priority level 0-3. If a priority level exceeds its maximum number of consecutive time slices, the task scheduler will give a time slice to a task at a lower priority, therefore, preventing lockouts of low priority tasks. The highest priority level is level 0; its weighting factor is requested as shown below.

PL 0 WT.FACTOR–

Table 9-1. GENTX Prompts

Request	Range	Default	Discussed in Paragraph
MEMORY AVAILABLE –	Note 7	none	9.3.3.1
LINE FREQ.	50 or 60	60	9.3.3.1
TIME SLICE	Note 1	6	9.3.3.1
PL 0 - 3 WT. FACTOR	Note 1	10	9.3.3.1
COMMON SIZE	Primarily limited by size of available memory.	0	9.3.3.1
# OF EXP CHASSIS	1-7	0	9.3.3.1
CHASSIS 1 - 4 INT LEVEL	3, 4, 6, 7 (990/4) 3, 4, 6 - 15 (990/10)	None None	9.3.3.1
CHASSIS 5 - 7	3, 4, 6, 7 (990/4) 3, 4, 6 - 15 (990/10)	None None	9.3.3.1
CHASSIS	0 - 7	None	9.3.3.2
DEV NAME	Note 2	None	9.3.3.2
DEV TYPE	Listed in table 7-3.	None	9.3.3.2
LEFT CASS/PTP NAME	Note 2	None	9.3.3.2
RIGHT CASS/PTR NAME	Note 2	None	9.3.3.2
STATION #	1 - 127	None	9.3.3.2
CRU BASE ADDR	0 - >1FFE	Device dependent	9.3.3.2
ACCESS MODE	FILE,RECORD, or CHARACTER	Listed in table 7-3.	9.3.3.2
INT LEVEL	3, 4, 6, 7 (990/10) 3, 4, 6 - 15 (990/10)	Device dependent	9.3.3.2

Table 9-1. GENTX Prompts (Continued

Request	Range	Default	Discussed in Paragraph
INT POSITION	0 - 15	None	9.3.3.2
TIMEOUT COUNT	Note 1	Device dependent	9.3.3.2
# OF DRIVES	1 - 4		9.3.3.2
CRU INT LINE	0 - 31	15	9.3.3.3
ENTRY LABEL OF DSR	Note 3	None	9.3.3.3
ENTRY LABEL OF ROUTINE	Note 3	None	9.3.3.3
INT BRANCH LABEL	Note 3	None	9.3.3.3
EXTENSION DATA	Note 3	None	9.3.3.3
SVC #	Note 4	None	9.3.3.4
ENTRY LABEL	Note 4	None	9.3.3.4
XOP #	1 - 14	None	9.3.3.5
WORKSPACE LABEL	Note 5	None	9.3.3.5
ENTRY LABEL	Note 5	None	9.3.3.5
TASK ID #	0 - 0F, 11 - FE	None	9.3.3.6
PRIORITY LEVEL	Note 6	None	9.3.3.6
INITIAL DATA LABEL	Note 6	None	9.3.3.6
MULTIPLE DYNAMIC TASKS (Y OR N)	Y or N	None	9.3.3.6
# OF DYNAMIC TASKS	1 - 256	None	9.3.3.6
# OF PROCEDURES	0 - 128	None	9.3.3.6
CONSOLE DEV NAME	Device name of a 733 ASR, 733 KSR, 911 VDT, or 913 VDT	None	9.3.3.7
DEFAULT DISC DEV	Note 8	None	9.3.3.7

Table 9-1. GENTX Prompts (Continued)

Request	Range	Default	Discussed in Paragraph
DEFAULT PRINT DEV	Any printing device	DUMY	9.3.3.7
ASSIGN LUNO	1 - F0	None	9.3.3.7
DEV NAME	Device name	None	9.3.3.7
# OF SPARE DEV LUNO BLOCKS	0 - 63	5	9.3.3.7
# OF SPARE FILE LUNO BLOCKS	0 - 63, Note 9	5	9.3.3.7
# OF FILE CONTROL BLOCKS	0 - 53, Note 9	3	9.3.3.7
# OF DEFAULT BUFFERS	1 - 20	0	9.3.3.7
BUFFER SIZE	2 - 1024	None	9.3.3.7
# OF GENERAL BUFFERS	1 - 50	0	9.3.3.7
BUFFER SIZE	1 - 1024	None	9.3.3.7
UPPER THRESHOLD –	1 - 99	None	9.3.3.8
LOWER THRESHOLD –	1 - 99	None	9.3.3.8

Notes:

1. Any positive value that would have practical significance is accepted. The actual limit 1 - 65,535.
2. Device names consist of one to four characters, the first of which must be alphabetic.
3. These inputs apply to user-supplied DSRs for non supported devices.
4. These inputs apply to user-supplied supervisor call routines.
5. These inputs apply to extended operation routines.
6. These inputs apply to tasks.
7. Memory available for use by GENTX.
8. Disc device name as it appeared in the name request.
9. Not prompted if no diskettes are defined in the system.

Table 9-2. System Timing Parameters

Parameter	Unit of Measurement	Use	How Specified
Real Time Clock Cycle	8.3 ms. for 60 Hz. 10 ms. for 50 Hz.	Used to derive System Time Unit and to define maximum time period for a time slice	Determined by power line frequency.
System Time Unit	Real Time Clock Cycles	Used to define Timeout Counts, Time Delays, and Do Not Suspend periods.	6 cycles (50 ms.) for 60 Hz. 4 cycles (40 ms.) for 50 Hz.
Maximum time period for a Time Slice	Real Time Clock Cycles	Used by Scheduler.	During system generation.
Timeout Counts	System Time Units	Used by IOS and DSR to detect device errors.	During system generation.
Time Delay	System Time Units	Used by Scheduler.	Time Delay supervisor call.
Do Not Suspend periods.	System Time Units	Used by Scheduler.	Do Not Suspend supervisor call.

The user enters the maximum number of consecutive time slices for priority level 0, followed by a carriage return. When a carriage return only is entered, GENTX uses the default value of 10. GENTX then requests a weighting factor for priority levels 1, 2, and 3 in a similar manner.

PL 1 WT.FACTOR–

PL 2 WT.FACTOR–

PL 3 WT.FACTOR–

The next request is as follows:

COMMON SIZE–

The user enters the number of bytes to be allocated for the COMMON area of memory, followed by a carriage return. The address and size of the COMMON area is supplied to a task in response to a Get COMMON Data Address supervisor call. If a TXDS system is being generated, enter at least 170. When the user enters a carriage return only, no COMMON area is allocated in memory.

The next request is as follows:

OF EXP CHASSIS –

The user enters the number of CRU expansion chassis connected in the system, a decimal value, 0 through 7, followed by a carriage return. When the user enters a carriage return only, GENTX uses the default value, 0. The response to this request causes GENTX to request interrupt level assignments for the expansion chassis. Chassis 1 through 4 share an interrupt level, and chassis 5 through 7 share another interrupt level. When the response to this request is 0, explicitly or by default, the next two requests are omitted.

When the response to the number of expansion chassis is greater than 0, GENTX prints the following request:

CHASSIS 1-4 INT LEVEL –

The user specifies the interrupt level for chassis number 1 through 4, a decimal value followed by a carriage return. When the response to the number of expansion chassis is less than 5, the next request is omitted.

When the response to the number of expansion chassis is greater than 4, GENTX prints the following request:

CHASSIS 5-7 INT LEVEL –

The user specifies the interrupt level for chassis number 5 through 7, a decimal value followed by a carriage return.

9.3.3.2 Supported Peripheral Device Definitions. The peripheral devices connected to the CRU through the CPU chassis are defined as a group. Each device, if it conforms to the standard interrupt interface configuration, may be placed on the same interrupt level as another device with like constraints. The standard interrupt interface configuration is that CRU bit 15 from the base address of the device will be "set" when an interrupt occurs. Table 9-3 shows which supported devices may share an interrupt level. The user must determine if a special device conforms to this requirement.

The peripheral devices connected to the CRU through each expansion chassis are also defined as a group. The devices on expansion chassis 1-4 share an interrupt level, and the devices on expansion chassis 5-7 share another interrupt level. Each device within an expansion chassis has a unique interrupt identifier for use in interrupt decoding.

Expansion chassis interrupts may also be shared with devices connected to the CRU through the CPU if the devices conform to the standard interface configuration.

The following request begins the peripheral-device definition portion of the definition phase:

CHASSIS –

The user enters a digit, 0 through 7, followed by a carriage return, to specify the group of devices to be defined.

A response of 0 specifies that the subsequent device definitions (until the next chassis request) are those of devices connected within the CPU. A response of 1 through 7 specifies that subsequent device definitions are those of devices connected through expansion chassis 1 through 7, respectively. The response to this request must be consistent with the response to the number of the expansion-chassis request described previously. A response of a carriage return only terminates the device-definition portion of the definition phase.

GENTX issues the following request to identify the first device to be defined in the chassis:

DEV NAME –

The user enters a user-defined device name, consisting of up to four characters, the first of which must be alphabetic, the rest may be alphanumeric. The device name is followed by a carriage return. GENTX then requests the appropriate device information as described in the following paragraphs. When the user enters a carriage return only, GENTX terminates the current chassis definition, and repeats the "CHASSIS –" request for the next chassis. If the user wishes to correct an invalid input of a previously defined device, he may enter the name of that device in response to the "DEV NAME –" prompt and reenter all the following requests for the device type and CRU base, etc. If a user is defining a disc drive, then it is necessary to make the name 3 characters long.

Following the request for a device name, GENTX issues the following request to identify the type of device. The type of device will determine the succeeding questions that will be asked.

DEV TYPE –

The user enters a device-type keyword, as listed in table 9-3, followed by a carriage return. If a carriage return only is entered, the device definition for the above requested device name is deleted.

Table 9-3. GENTX Device Keywords

Device Type Keyword	Device	Default Access Mode	Default Time-Out Count	Default CRU Base	Default Interrupt Level (CPU Only)	Multiple Devices per Interrupt Level
TTY	ASR 33 Teletype	RECORD	8192	0	6	Yes
ASR	733 ASR Electronic Data Terminal	RECORD	8192	0	6	Yes
KSR	733 or 743 KSR Electronic Data Terminal	RECORD	8192	0	6	Yes
LP	Line Printer	FILE	4096	60_{16}	6	Yes
FLP	2230 or 2260 Line Printer	FILE	4096	60_{16}	6	Yes
CR*	Card Reader	FILE	4096	40_{16}	4	No
V913	913 Video Display	RECORD	None	$C0_{16}$	3	Yes
COM	Communications Device	None	None	None	None	N/R
FD	Diskette Drive	None	None	80_{16}	7	N/R
V911	911 Video Display Terminal	RECORD	None	$C0_{16}$	3	Yes
SD	Special Device	RECORD	None	None	None	U/D

N/R - Not Recommended
U/D - User decision
* Card reader must have an exclusive interrupt assignment.

If the user enters one of the first eleven keywords in the table, the device is supported by TX990 Operating System. The last keyword, SD (special device) specifies a nonsupported device. Additional GENTX requests for information on nonsupported devices are described in paragraph "Defining Other Peripheral Devices".

When the user enters the keyword V911 or V913 as the device type, GENTX requests the following:

STATION # –

The user enters a decimal number, 1 through 127, followed by a carriage return. The number is the station number used in character mode I/O to the Video Display Terminal (VDT). There is no default for station number.

When the user enters the keyword ASR or TTY, GENTX prints the following requests: the name for the left cassette/paper-tape punch, and the right cassette/paper-tape reader:

LEFT CASS/PTP NAME –

RIGHT CASS/PTR NAME –

Following each request, the user enters a user-defined device name, one to four characters, the first of which must be alphabetic. The device name is followed by a carriage return. The user assigns the cassette or paper-tape units to LUNOs by means of these device names.

The next request for each TX990 supported device, except a communications device, is as follows:

CRU BASE ADDR –

The user enters the CRU base address as a hexadecimal value followed by a carriage return. A greater-than character (>) entered as the first character identifies the entry as hexadecimal in the range of numbers 0 through $1FFE_{16}$. The CRU base address is the CRU address shifted one bit position to the left (multiplied by 2), and must be an even number. On top of each chassis is a chart which defines the CRU base for each device. If a carriage return is entered, a standard base will be used. Defaults are listed in table 9-3.

The fourth request for each device, except diskette or a communications device, is as follows:

ACCESS MODE –

The user enters one of the following keywords, followed by a carriage return:

- FILE, for a file-oriented device, or for a 913 or 911 VDT that is to be used in either the file mode or the character mode,
- RECORD, for a record-oriented device, or for 913 or 911 VDT that is to be used in the record mode, as well as character mode (if a VDT is to be used as a system console, it must be in RECORD mode).
- CHAR, for a 913 or 911 VDT to be used in the character mode only.

When the user enters a carriage return only, the default access mode for the specified device applies. Default access mode for devices are listed in table 9-3.

One of the two formats is used for the fifth request, depending on whether the device is connected to the CRU through the CPU or through an expansion chassis. When the device is connected through the CPU (CHASSIS - 0), the request is as follows:

INT LEVEL –

The user enters a decimal or hexadecimal value followed by a carriage return. The entry represents the interrupt level for the device. If a carriage return only is entered, a standard level will be used. Defaults are listed in table 9-3. The user must not enter the interrupt level of the CLOCK. On a 990/4 computer the clock interrupt level is 5; on a 990/10 computer the clock interrupt level is 5 or 10. When the device is connected through an expansion chassis (chassis entry not equal to zero), the request is as follows:

INT POSITION –

The user enters a decimal value followed by a carriage return. The entry represents the interrupt identifier returned by the expansion chassis when a device generates an interrupt. The interrupt identifier is a number in the range of 0 through 15 and is determined by connections within the CRU expansion chassis.

The sixth request for each device, except a diskette or a communications device, is as follows:

TIME-OUT COUNT –

The user enters a decimal value followed by a carriage return. The value represents the number of system time units to be used as a timeout count for the device. This timeout count will be used by the DSR to determine how long to wait on a device before determining that a timeout error has occurred. When the user enters a carriage return only, the default value for the specified device applies. Default values for devices are listed in table 9-3. Entering a zero specifies no timeout.

If the device is a diskette the following request is made instead of the timeout count request:

OF DRIVES –

The user must enter a number in the range of one to four.

If the user wishes to define more than one diskette drive he enters the first name and a 2, 3, 4 is concatenated onto the first three characters of the disc name up to the number entered for # OF DRIVES– prompt, thus all drives are defined from the first name entered. The names that were generated may not be used again, nor may they have already been used.

If the user wants to generate a system with more than one diskette controller, he can do so by entering a different three-character name so that the names generated by concatenating a 2, 3, 4 are different than the disc names that have already been defined.

When the device is a supported device, GENTX next repeats the device name request to define another device.

9.3.3.3 **Special Device Definition.** Special devices are those which are not supported by TI supplied modules. When the device type request is printed, the user may enter the keyword SD.GENTX requests the same six parameters as for supported devices, then prints the following request:

CRU INT LINE –

The user enters a number in the range of zero through 31, followed by a carriage return. The number represents the displacement from the base address to the CRU line that is set when an interrupt occurs.

The next request for a nonsupported device is as follows:

ENTRY LABEL OF DSR –

The user enters the entry label of the Device Service Routine (DSR) for the device followed by a carriage return. The entry point required in response to this request is the entry point for supervisor call processing. The user supplies the DSR for the device when he links the TX990 system together.

The next request for a nonsupported device is as follows:

ENTRY LABEL OF ROUTINE –

The user enters the entry label of the interrupt routine for the device, usually a part of the DSR, followed by a carriage return.

The next request for a nonsupported device is as follows:

INT BRANCH LABEL –

The user enters the label of the interrupt branch followed by a carriage return. Typically an interrupt routine performs processing common to any interrupt for the device, then branches to perform processing appropriate to the context of the interrupt. For example, an unsolicited interrupt usually requires different processing than the interrupts that occur while performing an I/O operation specified in a supervisor call. The label to which the unsolicited interrupt causes the system to branch is supplied in response to the above request.

The next request for a nonsupported device is as follows:

EXTENSION DATA –

The user enters a source statement in assembly language, followed by a carriage return. The user may enter an exclamation point (!) following each field of the source statement to tab to the beginning of the next field. Specifically, entering an exclamation point positions the following character at position 8, 13, or 31.

The source statement will be entered in the source file for TXDATA following the source statements supplied by GENTX for the PDT for the device. GENTX repeats the request until the user enters a carriage return only, which terminates the definition for the device. Subsequent source statements are placed following the initial statement. The group of source statements entered in response to this series of requests forms an extension to the PDT that contains device-related data. When the user terminates the definition of the device, GENTX repeats the device name request to define another device.

9.3.3.4 Defining User-Supplied Supervisor Calls. When the user has terminated the peripheral-device portion of the definition phase, GENTX prints the following request:

SVC # –

The user enters a hexadecimal number, followed by a carriage return. The number represents the call code of a user-supplied supervisor call (XOP level 15). The valid codes begin at 80_{16} and extend as far as required to include all user-supplied calls. For example, if three supervisor calls are supplied by a user, the call codes must be 80_{16}, 81_{16}, and 82_{16}. They may be entered in any order, but must all be included. Similarly, if a single supervisor call were supplied, its call code would be 80_{16}. When the user enters a carriage return only, GENTX omits the next request and terminates the supervisor-call portion of the definition phase.

When the user has entered a supervisor-call code, GENTX prints the following request:

ENTRY LABEL –

The user enters the entry label of the supervisor-call code, followed by a carriage return. The user must supply the supervisor-call routine when linking the TX990 operating system. GENTX then repeats the SVC number request.

9.3.3.5 Defining Extended Operations. When the user has terminated the supervisor call portion of the definition phase, GENTX enters the extended operation portion of the definition phase and prints the following request:

XOP # –

The user enters a decimal number, 0 through 14, followed by a carriage return. The number represents the level of a software implemented extended operation. The user supplies an XOP routine for the extended operation. When the user enters a carriage return only, GENTX omits the next two requests and terminates the extended operation portion of the definition phase. For further details concerning extended operation routines refer to Appendix C.

When the user enters an XOP number, GENTX prints the following request:

WORKSPACE LABEL –

The user enters the label of the workspace of the XOP routine, followed by a carriage return. GENTX then prints another request:

ENTRY LABEL –

The user enters the label of the entry to the XOP routine, followed by a carriage return.

The user must supply the object programs that process the XOP when linking the TX990 Operating System. GENTX then repeats the XOP number request.

9.3.3.6 Defining Tasks. When the user terminates the extended operation portion, GENTX enters the task definition portion. During this phase of system definition, GENTX adds to the internal table of data from which it will generate the source statements for TASKDF. Both system tasks, and user-supplied tasks must be defined at this time if they are to be linked with the new TX990 Operating System. Table 9-4 shows the task IDs for the system tasks under the "ID" column, the priority level of each task is under the "PRIORITY" column, and the label on the first word of each task is under the "DATA LABEL" column. Each system task, and its purpose, is described in Appendix J.

Table 9-4. System Task Definition

Task	ID	Priority	Data Label
FMP1	>F0	0	FMP1
FMP2	>F1	0	FMP2
FMP3	>F2	0	FMP3
FMP4	>F3	0	FMP4
FUR	>B	1	FUR
VOLUME	>C	0	VOLUME
DTASK	>D	1	DIAGTS
OCP	>F	1	OCP
CNTROL	>16	1	CNTROL
STASK	>10	1	STRT

The user may select a task ID for his tasks but they must not be the same ID as a system task ID. Task IDs $1\text{-}F_{16}$ and FO_{16} $-FF_{16}$ are reserved for system tasks. The task ID will be used to bid the task, or to execute the task from OCP. The priority level will define which tasks will use the most time slices. Priority level zero is the highest available priority level. The data label is a defined (using the assembler directive, DEF) label on the first three words of a task. These words define the initial Workspace, Program Counter, and End Action address. A reference (an assembler REF directive) will be generated inside the TASKDF module, so that the TX990 Operating System will have the information needed to begin execution of that task.

The following requests are made by GENTX asking for the above information.

TASK ID # –

The user enters a number, followed by a carriage return. The number represents the task identifier by which the task is identified to the system, and must be less than FF_{16}. Identifier 10_{16} is assigned to the dynamic-task area, and user tasks should be assigned identifiers greater than 10_{16}. When the user enters a carriage return only, GENTX terminates the task-definition portion of the definition phase, and begins requesting information for assigning LUNOs.

For each task, GENTX prints two requests following entry of the task identifier. The first request is as follows:

PRIORITY LEVEL –

The user enters a two-digit hexadecimal number, followed by a carriage return. The first digit must be one of the digits listed in table 9-5, which also lists the meaning of each digit. When the digit that applies to the task is zero, it may be omitted. The second digit is the actual priority level. Priority level 0 should be used only for system tasks.

The second request is as follows:

INITIAL DATA LABEL –

The user enters the label on the first word in the task, followed by a carriage return. Then GENTX repeats the task ID request. When all tasks have been defined and the user has terminated the task-definition portion, GENTX asks if the system is to support multiple dynamic tasks.

Table 9-5. Priority Digits

Digit (Hexadecimal)	Meaning
0	Privileged; not active at initial load of operating system, restart, or power restart.
1	Privileged; active at initial load of operating system or restart; not active at power restart.
2	Privileged; not active at initial load of operating system or restart; active at power restart.
3	Privileged; active at initial load of operating system, restart, or power restart.

The request is as follows:

MULTIPLE DYNAMIC TASKS (Y OR N) –

The user should enter Y or N. If the response is N, GENTX proceeds to the Assign LUNO phase. If the response is Y, GENTX requests the maximum number of dynamic tasks allowed to be in the dynamic task area at one time, as follows:

OF DYNAMIC TASKS –

The user enters a decimal number between 1 and 255, inclusive. Then GENTX requests the maximum number of procedures allowed to be installed in the dynamic task area at one time, as follows:

OF PROCEDURES –

The user enters a decimal number between 0 and 127, exclusive. GENTX then proceeds to the Assign LUNO phase.

9.3.3.7 **Information For Assigning LUNOs.** The following information is used to generate: names for LUNO assignments; memory blocks that will be reserved for device LUNOs; and file LUNOs and memory blocks that will be reserved for communication devices.

The following request is made by GENTX asking for the device name of the system console. Most of the utilities, and the Operator Communication Package (OCP) interact with the user through the system console. The request is as follows:

CONSOLE DEV NAME –

The user enters the device name (previously assigned) for the system console, followed by a carriage return. The device must be either device type ASR, KSR, or VDT. When the device is device type VDT, it must have been previously defined for the record mode. The user may enter a carriage return only to specify no system console.

GENTX makes the following request:

DEFAULT DISC DEV –

The user must enter the diskette device name (previously defined) as it appeared in the name request. The user may enter a carriage return to specify no default disc. The default disc name is used when a utility is referring to the system disc; it is also used when no disc name is appended to the pathname of a file.

GENTX makes the following request:

DEFAULT PRINT DEV –

The user enters the device name (previously defined) for the system default printers. The device can be any printing device. The user may enter a carriage return to specify the Dumy device. The device is used by some of the TI-supplied utilities when no listing device is specified. It is a required parameter in a TX990 Operating System that supports a Terminal Executive Development System.

Next, GENTX prints the following request:

ASSIGN LUNO –

The user enters a hexadecimal value, followed by a carriage return. The range of values is 1 through $F0_{16}$. The value represents a LUNO by which a device is accessed for I/O operations. When the user enters a carriage return only, GENTX skips the next request of the definition phase. Otherwise, GENTX requests the name of the device to be assigned to the LUNO, as follows:

DEV NAME –

The user enters the device name previously assigned to a device. GENTX then repeats the assign LUNO request.

The next request is as follows:

OF SPARE DEV LUNO BLOCKS –

The user enters a decimal number followed by a carriage return. The number represents the number of device logical unit blocks to be provided in the system in addition to those numbers assigned previously. When the ALUNO OCP command of the File Utility supervisor call is included in a TX990 system, there must be logical unit blocks for all LUNOs that will be assigned at one time. When the user enters a carriage return only in response to the request, GENTX provides five additional blocks. In systems which support multiple dynamic tasks, more blocks may be necessary.

The next request, if a diskette has been defined, is as follows:

OF SPARE FILE LUNO BLOCKS –

The user enters a decimal number followed by a carriage return. The number represents the number of file logical unit blocks to be provided in the system. There must be file logical unit blocks for all LUNOs that will be assigned to files at one time. When the user enters a carriage return, GENTX provides five additional blocks.

The next request, if a diskette has been defined, is as follows:

OF FILE CONTROL BLOCKS –

The user enters a number between 0 and 50. The number represents the maximum number of files that can be open at any time. If File Management is to be used with TXDS, a minimum of three File Control Blocks are required.

The next request is as follows:

OF DEFAULT BUFFERS –

The user enters a decimal number from 0 to 20. If a carriage return is entered, no buffers are reserved and GENTX skips the size request. If GENTX accepts the number, it requests the size of the buffer in bytes by presenting the following prompt:

BUFFER SIZE –

The user must enter a decimal number from two to 1024.

The next request is:

OF GENERAL BUFFERS –

The user enters a decimal number from 0 to 50. If a carriage return is entered, no buffers are allocated and GENTX omits the next request and terminates the general buffer portion of the definition phase.

If GENTX accepts the number it requests:

BUFFER SIZE –

The user enters a decimal number from 1 to 1024 which represents the number of bytes in each buffer. GENTX then repeats the GENERAL BUFFER request. The default and general buffers are simply reserved areas in the system table area which may be accessed by user tasks via the Get Data and Put Data SVCs.

9.3.3.8 **Communication Threshold Definition.** If a device name was defined as a communications device by entering "COM" for the "DEV TYPE–" request, GENTX requests the communications thresholds as follows:

UPPER THRESHOLD –

LOWER THRESHOLD –

(The user is referred to the Communications Systems Manual for a detailed explanation of thresholds.) The user enters decimal numbers, from 1 to 99, and the upper threshold must be greater than or equal to the lower. If in error, the sequence will be repeated.

9.3.3.9 **Suggestions For Defining Devices.** The following request of GENTX are repeated until the user enters a carriage return only in response. This allows an indefinite number of each item to be defined, as appropriate for the application. These requests are as follows:

CHASSIS

SVC#

XOP#

TASK ID#

ASSIGN LUNO

OF GENERAL BUFFERS

When the same chassis number is entered by the user in response to a subsequent chassis request, additional devices entered following the second entry of the chassis number are added to the group originally defined. This allows definition of devices by function or by physical arrangement rather than by connection within the computer. That is, in the following example, the line printer and card reader are connected through the CPU, and the VDT is connected through expansion chassis 1:

```
CHASSIS – 0

    DEV NAME – LIPR
    DEV TYPE – LP
        .
        .
        .

    DEV NAME –

CHASSIS – 1

    DEV NAME – GTTY
    DEV TYPE – V913
        .
        .
        .

    DEV NAME –
CHASSIS – 0

    DEV NAME – CRDR
    DEV TYPE – CR
```

When the same device name, supervisor call number, extended operation number, task identifier, or LUNO is entered again, the previously entered information is replaced by the new information, redefining the item. To delete a device name, enter the chassis number and the device name. Then enter a carriage return only in response to the device type request.

9.3.4 **CONSTRUCTION PHASE.** After the last request of the Definition Phase of GENTX, the Construction Phase is entered. During the construction phase, GENTX generates source statements for TXDATA and TASKDF using the responses entered by the user during the definition phase. GENTX prints the following request:

TASKDF OUTPUT FILE NAME –

The user may enter the full file name or device name. If the file does not exist it will be created.

The user may inhibit output of TASKDF by entering a carriage return only. GENTX then prints the final request as follows:

TXDATA OUTPUT FILE NAME –

The user enters the full file or device name. When the file does not exist it will be created. The user may inhibit output of TXDATA by entering a carriage return only. GENTX then terminates, bidding the rebid task.

NOTE

DX10 output file names must be full pathnames; synonyms are not accepted.

9.3.5 **GENTX ERROR MESSAGES.** Table 9-6 lists the error messages or warnings produced by GENTX.

Table 9-6. GENTX Error Messages

Error	Reason	Recovery
INVALID INPUT	Response not valid.	Reenter response.
INPUT ERROR	Response not within range.	Reenter response.
MULTIPLY DEFINED INPUT	The input has already been previously defined.	Reenter response.
LUNO 0 ASSIGNED TO CONSOLE	User tried to assign LUNO 0 to a device other than the console.	Select a LUNO other than 0. Reenter response.
INVALID DEV	The device type name was not valid	See table 7-3. Reenter response.
ASSIGN ERROR	Could not assign LUNO 10 to the output device or file.	Release some LUNOs and re-execute the task; or an invalid device name was entered. Re-enter response.
OPEN ERROR	Could not open output device or file.	
CLOSE ERROR	Could not close, or release output device of file.	
STORAGE ALLOC. ERROR	GENTX table area exhausted. Memory area not large enough.	Reexecute task and increase size of the MEMORY AVAILABLE request.

9.4 ASSEMBLING THE SOURCE MODULES

After termination of GENTX, the user must assemble the two source modules, TXDATA and TASKDF. Assembly is accomplished by executing the TXDS Assembler, TXMIRA. Each source module is assembled separately.

For detailed instructions for operating TXMIRA consult the TXDS Programmer's Guide.

9.5 LINKING THE OBJECT MODULES

After assembling the two modules it is necessary to use the Object Manager to select the TI-supplied object modules that support features of the TX990 operating system. Selection of these features is made from the object module files residing on the diskette which contains the TX990 parts and an object file residing on the system diskette. The files are described below:

- :DSRLIB/OBJ – Object file that contains DSR modules supplied by Texas Instruments;
- :DYNTSK/OBJ – Object file that contains modules to support multiple dynamic tasks;
- :OCPLIB/OBJ – Object file that contains OCP modules supplied by Texas Instruments;
- :FMPLIB/OBJ - Object file that contains FILE MANAGEMENT modules supplied by Texas Instruments;
- :CNTROL/OBJ – Object file that contains the TXDS control program;
- :TXLIB/OBJ – Object file that contains TX990 modules supplied by Texas Instruments.

The user must create a file that will contain all the object modules he wishes to link to form the new operating system. This new file will contain modules from each of the object library files. The user should refer to table 9-7 to select the modules which support the desired features. The modules should be selected from the files in the following order:

:DSRLIB/OBJ

:DYNTSK/OBJ

:OCPLIB/OBJ

:FMPLIB/OBJ

:CNTROL/OBJ

:TXLIB/OBJ

Table 9-7. TX990 Operating System Modules

Name	Function	Approximate Size (Bytes)	Required	Required for OCP	Required for File Management	List In TASKDF	Notes
TXDATA	System configuration information, interrupt decoders, trap initialization data.	Typical 600-1200	X				
TASKDF	Task Definition module. All tasks must be identified here.	Typical 100-200	X				
				DSRLIB			
FPYDSR	FD800 Diskette support	1066			X		TXDATA must contain Device Name Table (DNT)** and Physical Device Table (PDT)** entries for diskettes.
***FLPDSR	Models 2230 and 2260 Line Printer Support	272					TXDATA must contain DNT** and PDT** entries for Line Printers.
DSR733	733 ASR Keyboard/ Printer and Cassette Unit support	1460		*			TXDATA must contain DNT** and PDT** entries for 733 ASRs.
DSR913	913 VDT support for record or file mode I/O	700		*			TXDATA must contain DNT** and PDT** entries for VDTs. System must include STA913, unless only character mode I/O is used.
DSR911	911 VDT support as a sequential I/O device	730		*			TXDATA must include DNT** and PDT** entries for VDTs. System must include STA911.

* One of the DSRs with an asterisk must be included for OCP.
** DNTs and PDTs are placed in TXDATA by using the GENTX utility.
*** FLPDSR and LPDSR must not be included in the system together.

Table 9-7. TX990 Operating System Modules (Continued)

Name	Function	Approximate Size (Bytes)	Required	Required for OCP	Required for File Management	List In TASKDF	Notes
***LPDSR	Models 306, 588, and 810 Line Printer support	264					TXDATA must contain DNT** and PDT** entries for Line Printers.
CRDSR	Card Reader support	508					TXDATA must contain DNT** and PDT** entries for Card Readers.
KSRDSR	733 or 743 KSR Keyboard/Printer support	700		*			TXDATA must contain DNT** and PDT** entries for 733 KSRs.
DSRTTY	ASR33 Teletype support	1460		*			TXDATA must contain DNT** and PDT** entries for ASRs.
DSR5MT	5MT/6MT special device support	532					DSR module to control the 5MT/6MT I/O Interface.
DIGDSR	32-IN Transition Detection Module Special device support	177					DSR module to control the 32-IN Transition Detection module.
			OCPLIB				
OCPTSK	OCP data base	250		X		X	System must include TSKFUN, IOSUPR, CNVRSN, FURSVC, OCPTBL, OCPPRC, and OCPEND.
OCPTBL	OCP configuration tables	180		X			System must include TSKFUN IOSUPR, CNVRSN, FURSVC, OCPTSK, OCPPRC, and OCPEND.

* One of the DSRs with an asterisk must be included for OCP.
** DNTs and PDTs are placed in TXDATA by using the GENTX utility.
*** FLPDSR and LPDSR must not be included in the system together.

Table 9-7. TX990 Operating System Modules (Continued)

Name	Function	Approximate Size (Bytes)	Required	Required for OCP	Required for File Management	List In TASKDF	Notes
OCPPRC	OCP main procedure	600		X			System must include TSKFUN, IOSUPR, CNVRSN, FURSVC, OCPTSK, OCPTBL, and OCPEND.
OCPIOU	STASK, SIO, REWIND, FSPACE, and BSPACE OCP commands	600					System must include TSKFUN, IOSUPR, CNVRSN, FURSVC, OCPTSK, OCPTBL, OCPPRC, and OCPEND.
DOCPIOU	SPROC, STASK, SIO, REWIND, FSPACE, and BSPACE OCP commands	790					This module must be used in place of OCPIOU if multiple dynamic tasks are allowed; it is in :DYNTSK/OBJ.
OCPTLD	DWKSP, KTASK, and KIO OCP commands	330					System must include TSKFUN, IOSUPR, CNVRSN, FURSVC, OCPTSK, OCPTBL, OCPPRC, OCPSLD, and OCPEND.
OCPSLD	LMEM, DMEM, SBKPT, CBKPT, ADD, SUB, and JMP OCP commands	550					System must include TSKFUN, IOSUPR, CNVRSN, FURSVC, OCPTSK, OCPTBL, OCPPRC, and OCPEND.
OCPTAD	TIME and IDATE OCP commands	520					System must include TSKFUN, IOSUPR, CNVRSN, FURSVC, OCPTSK, OCPTBL, OCPPRC, and OCPEND.

Table 9-7. TX990 Operating System Modules (Continued)

Name	Function	Approximate Size (Bytes)	Required	Required for OCP	Required for File Management	List In TASKDF	Notes
OCPLRT	ALUNO, RLUNO, LPROG, and EXECUTE OCP commands	420					System must include TSKFUN, IOSUPR, CNVRSN, FURSVC, OCPTSK, OCPTBL, OCPPRC, OCPEND, and TSKLDR.
DOCPLRT	AL, RL, LP, EX, IT, DT, IP, DP, OCP commands	700					Must be included to support multiple dynamic tasks. This module is in :DYNTSK/OBJ.
OCPEND	OCP dummy external reference/definition module	20		X			System must include TSKFUN, IOSUPR, CNVRSN, FURSVC, OCPTSK, OCPTBL, and OCPPRC.
				CONTROL			
CONTROL	TXDS control program	1340				X	
				FMPLIB			
TXFMP1	File management data section for drive 1	100			X	X	
TXFMP2	File management data section for drive 2	100			X†	X†	
TXFMP3	File management data section for drive 3	100			X†	X†	
TXFMP4	File management data section for drive 4	100			X†	X†	

†Required only if the corresponding drive is included in the system.

Table 9-7. TX990 Operating System Modules (Continued)

Name	Function	Approximate Size (Bytes)	Required	Required for OCP	Required for File Management	List In TASKDF	Notes
TXFMP	File Management main drive and opcode decoder	266			X		
FMOPEN	Open processor for files	500			X		
FMCLOS	Close processor for files	336			X		
FMREAD	Read processor for files	520			X		
FMWRIT	Write processor for files	560			X		
FMFBSP	Forward and backspace processors for files	420			X		
FMUTIL	Utility routines to support the other file management opcode processors	500			X		
FURTSK	File utility opcode decode for opcodes 90_{16} to 99_{16}	124			X	X	
FURSVC	Contains assign and release LUNOs for files and devices including those for supervisor call 15_{16}	700			X		If only assign and release LUNO support is required the user may include only this module without the rest of File Management. The system must include IOSUPR and TSKFUN.

Table 9-7. TX990 Operating System Modules (Continued)

Name	Function	Approximate Size (Bytes)	Required	Required for OCP	Required for File Management	List In TASKDF	Notes
SERDIR	Searches the file directory for a particular file	154			X		
ALUNIT	Allocates AUs on the diskette, also deallocates AUs on the diskette	292			X		
FILESV	Contains the create, delete, and compress modules for file utility	278			X		
CHSVC	Contains the change protection, and change name modules for file utility	276					
				TXLIB			
TXROOT	Scheduler, clock handler, internal interrupt processors, interrupt and XOP return processing, queueing routines, time delay management, date/time control, error handling, bid task logic.	1000	X				
SVC913	913 VDT Utility supervisor call routine	430					
STA913	913 VDT character mode input support	204					TXDATA must contain Keyboard Status Block (KSB) entries for VDTs.

Table 9-7. TX990 Operating System Modules (Continued)

Name	Function	Approximate Size (Bytes)	Required	Required for OCP	Required for File Management	List In TASKDF	Notes
SVC911	911 VDT Utility supervisor call	430					
STA911	911 VDT character mode input support	204					TXDATA must contain the Keyboard Status Block (KSB) entries for VDTs.
CRTPRO	Decodes the VDT 913, or VDT 911 supervisor call	164					Must be present if STA911, SVC911, STA913, or SVC913 modules are included
IOSUPR	Input/Output supervisor call routine	900		X	X		System must include TSKFUN.
MEMSVC	Get/Return Memory and Set Condition	56					
DMEMSVC	Same as MEMSVC	176					Supports multiple dynamic tasks. This module is in :DYNTSK/OBJ.
TBUFMG	Buffer management routine	176	X				
CNVRSN	Hexadecimal/decimal/binary conversion supervisor call routines	360		X	X		
EVENTK	Break key	250					Used in AMPL systems.
TSKLDR	Object module loader	462					System must include TSKFUN and IOSUPR.

Table 9-7. TX990 Operating System Modules (Continued)

Name	Function	Approximate Size (Bytes)	Required	Required for OCP	Required for File Management	List In TASKDF	Notes
DTSKLDR	Object module loader	520					Supports multiple dynamic tasks. This module is in :DYNTSK/OBJ.
IMGLDR	Program image loader	478					Loads images from program files.
DYNTSK	Multiple dynamic task support	780					This module is in :DYNTSK/OBJ.
TSKFUN	Task support function: Bid Task, Get Parameters, End of Task, Do Not Suspend, Time Delay, Change Priority, Unconditional Wait, Activate Suspended Task, Date and Time, and Get Common Address supervisor call routines	290					
TITTCM	GETDAT/PUTDAT Supervisor calls routine	140					
DTASK	Diagnostic task	242				X	System must include TSKFUN and IOSUPR.
TXSTRT	Initial and manual restart logic	200	X				May optionally be placed following module TXEND to provide more memory for dynamic task area. When TXSTRT is placed following TXEND, TXSTRT executes only on IPL, and manual restart is not possible.

Table 9-7. TX990 Operating System Modules (Continued)

Name	Function	Approximate Size (Bytes)	Required	Required for OCP	Required for File Management	List In TASKDF	Notes
TXEND	TX990 dummy external reference/definition module	12	X				
STASK	Initial startup task	180					May optionally be placed between TXSTRT and TXEND to be executed both during an IPL and during a manual restart. STASK begins execution at the completion of each IPL to identify the revision level of the operating system and to indicate that the IPL has executed successfully. When STASK is linked following TXEND, it is located in the dynamic task area, where it will be overlaid by the first user task installed in the dynamic task area. It is assigned task identifier 10_{16}. When it is desired to execute STASK following a manual restart, STASK must be linked ahead of TXEND.

There is no defined order in which modules must be included, although user-supplied object modules and programs must be included before the object module TXEND, in file: TXLIB/OBJ. It is logical to place a DSR with the other DSRs, a supervisor call routine with the supervisor call routines, and a task with the other tasks.

After the file has been generated by the Object Manager (OBJMGR), the user must link it with the two object modules, TXDATA and TASKDF. TXDATA must be first in the link sequence and TASKDF must be second.

Because many modules which may be incorporated in a customized TX990 system are optional, the modules TXEND and OCPEND contain many dummy symbol definitions. When optional modules are included, the link editing process results in multiply defined symbol errors, as shown in the following:

```
SYMBOL MULTIPLY DEFINED *W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*
SYMBOL = ALRSET   , MODULES = 47, 49

SYMBOL MULTIPLY DEFINED *W*W*W*W*W*W*W*W*W*W*W*W*W*W**W*W*
SYMBOL = BADID    MODULES = 42, 49
```

Such errors do not result in an incorrect link edit. To be sure that the multiply defined symbols are not caused by fatal errors, compare the second number given after MODULES in the error message to the module numbers given in the link map for: OCPLRT, OCPIOU, OCPEND, or TXEND.

The following is an example of part of such a link map:

OCPTAD	48	657A	01F8	INCLUDE	01/14/77	14:07:24	SDSMAC
OCPEND	49	6772	0016	INCLUDE	09/15/77	11:26:28	SDSMAC
TXEND	50	6788	0142	INCLUDE	10/05/77	13:15:42	SDSMAC

If the numbers match, the error is nonfatal and is to be expected.

After the files have been linked to form a new TX990 operating system, the new object module file containing the new TX990 must be defined as the system file by using the System Utility (SYSUTL) command "SF". Also, the TXBOOT program must be copied to the diskette by using the SYSUTL command "BC".

9.6 EXAMPLE OF SYSTEM GENERATION

The materials needed are:

- The TX990 parts diskette (diskette #1)
- The TXDS system diskette (one of TXDS system diskettes #2, 3, 6)
- A scratch disc on which the system can be built.

NOTE

Operator responses to prompts presented on the system console are underlined in the following procedure. Prompts are not underlined.

1. Load diskette #1 (TXPARTS) in diskette drive 2 (right). Load diskette #2, #3, or #6 (TXDS system) in diskette drive 1 (left).

2. Load the TX990 Operating System into memory by pressing the following keys on the Programmer's Panel.

   a. Halt

   b. Reset

   c. Load

   After the TX990 Operating System is loaded into memory the following message will be printed out or displayed on the system console:

   TX990 SYSTEM Release 2.2

   MEMORY SIZE (WORDS): 24576 MEMORY AVAILABLE: 16749

3. Bid TXDS by entering an exclamation point (!) on the terminal. The following display will appear on the system console:

   ```
   !
   TXDS 936215 ** 1/ 0 0: 0
   PROGRAM:
   ```

4. Load a scratch diskette in drive #1.

5. Initialize the scratch diskette by responding to the prompts printed out or displayed on the system console as follows:

   ```
   TXDS 936215 ** 1/0 0:02

   PROGRAM: :BACKUP/SYS*

   BACKUP AND INITIALIZE UTILITY 936212 *A

   OUTPUT DISC OR VOLUME NAME? DSC
   THE OUTPUT DISC MUST BE INITIALIZED
   INITIALIZE DSC ?(Y/N) Y
   DISC I. D. ? SCRATCH
   VOLUME NAME ? SCR

   CHECKING DSC
   ```

6. After the diskette has been initialized, the user must bid the System Generation (GENTX) Utility by responding to the prompts printed out or displayed on the system console as presented below. When no response to a prompt is indicated, the user must enter a carriage return/NEW LINE to cause the default-substitute to be used.

```
TXDS 936215 ** 1/ 0 0:12

PROGRAM:  :GENTX/SYS*

TX990 SYSTEM GENERATION 945673 *B

MEMORY AVAILABLE – 2000

LINE FREQ. –                          Take default (table 7-7)
TIME SLICE –                          Take default (table 7-7)
PL 0 WT. FACTOR –                     Take default (table 7-7)
PL 1 WT. FACTOR –                     Take default (table 7-7)
PL 2 WT. FACTOR –                     Take default (table 7-7)
PL 3 WT. FACTOR –                     Take default (table 7-7)
COMMON SIZE – 170
# OF EXP CHASSIS –
CHASSIS – 0
```

a. For ASR System

```
DEV NAME – LOG
DEV TYPE – ASR
LEFT CASS/PTP NAME – CS1
RIGHT CASS/PTR NAME – CS2
CRU BASE ADDR –                       Take default (table 7-3)
ACCESS MODE –                         Take default (table 7-3)
INT LEVEL –                           Take default (table 7-3)
TIME-OUT COUNT –                      Take default (table 7-3)
```

b. For 911 System:

```
DEV NAME – LOG
STATION # – 1
CRU BASE ADDR –                       Take default (table 7-3)
ACCESS MODE –                         Take default (table 7-3)
INT LEVEL –                           Take default (table 7-3)
TIME-OUT COUNT –                      Take default (table 7-3)
```

c. For 913 System:

```
DEV NAME – LOG
STATION # – 1
CRU BASE ADDR –                       Take default (table 7-3)
ACCESS MODE –                         Take default (table 7-3)
INT LEVEL –                           Take default (table 7-3)
TIME-OUT COUNT –
```

d. Resume system generation:

DEV NAME – DSC	Generate floppy
DEV TYPE – FD	
CRU BASE ADDR –	Take default (table 7-3)
INT LEVEL –	Take default (table 7-3)
# OF DRIVES – 2	Two drives: DSC and DSC2
DEV NAME – LP	
DEV TYPE – LP	
CRU BASE ADDR –	Take default (table 7-3)
ACCESS MODE –	Take default (table 7-3)
INT LEVEL –	6 on 911 system or take default (table 7-3)
TIME-OUT COUNT –	Take default (table 7-3)
DEV NAME –	Terminate sequence
CHASSIS –	
SVC # –	
XOP # –	
TASK ID # – >F0	FMP1
PRIORITY LEVEL – 0	
INITIAL DATA LABEL – FMP1	
TASK ID # – >F1	FMP2
PRIORITY LEVEL – 0	
INITIAL DATA LABEL – FMP2	
TASK ID # – >B	FUR
PRIORITY LEVEL – 1	
INITIAL DATA LABEL – FUR	
TASK ID# – >C	Volume name support
PRIORITY LEVEL – 0	
INITIAL DATA LABEL – VOLUME	
TASK ID # – >D	Diagnostic Task
PRIORITY LEVEL – 1	
INITIAL DATA LABEL – DIAGTS	
TASK ID # – >F	Add in OCP
PRIORITY LEVEL – 1	
INITIAL DATA LABEL – OCP	
TASK ID # – >16	TXDS CONTROL
PRIORITY LEVEL – 1	
INITIAL DATA LABEL – CNTROL	
TASK ID # –	Terminate sequence
MULTIPLE DYNAMIC TASKS (Y OR N) – N	
CONSOLE DEV NAME – LOG	
DEFAULT DISC DEV – DSC	
DEFAULT PRINT DEV – LP	
ASSIGN LUNO – 1	
DEV NAME – LOG	
ASSIGN LUNO –	Terminate sequence
# OF SPARE DEV LUNO BLKS –	Take default (table 7-3)
# OF SPARE FILE LUNO BLKS –	Take default (table 7-3)

```
# OF FILE CONTROL BLOCKS – 3
# OF DEFAULT BUFFERS –                          None
# OF GENERAL BUFFERS –
# OF GENERAL BUFFERS –                          Terminate sequence
TASKDF OUTPUT FILE NAME – :TASKDF/SRC           Note 1
TXDATA OUTPUT FILE NAME – :TXDATA/SRC           Note 2

END TX990 SYSGEN
```

7. Select the TI-supplied object modules to support the desired TX990 Operating System features and copy them to the scratch diskette by responding to the prompts as follows:

```
TXDS  936215  *A  1/0  00:00

PROGRAM:   :OBJMGR/*
990 OBJECT MANAGER  939870  **

     OUTPUT FILE:  DSC:TXPART/OBJ
       INPUT FILE:  DSC2:DSRLIB/OBJ
     REWIND INPUT FILE?  Y
FPYDSR    ?    C
DSR733    ?    C                                S if no ASR733
KSRDSR    ?    S
LPDSR     ?    C
FLPDSR    ?    S
DSR913    ?    S                                C if 913 VDT
CRDSR     ?    S
DSRTTY    ?    S
DSR911    ?    S                                C if 911 VDT
DSR5MT    ?    S
DIGDSR    ?    S
END-OF-FILE
```

Notes:

1. Constructs a source program for the task definitions and puts them in the file :TASKDF/SRC.

2. Constructs a source program for the TX990 Operating System data structures and puts them in the file :TXDATA/SRC.

```
     INPUT FILE: DSC2:OCPLIB/OBJ
   REWIND INPUT FILE? Y
OCPTSK    ?    A                    All of OCP is included,
END-OF-FILE                         without multiple dy-
                                    namic task support
     INPUT FILE: DSC2:FMPLIB/OBJ
   REWIND INPUT FILE? Y
VOLUME    ?    C
TXFMP1    ?    C
TXFMP2    ?    C
TXFMP3    ?    S                    No DSC3
TXFMP4    ?    S                    No DSC4
TXFMP     ?    C                    Rest of file
FMOPEN    ?    A
END-OF-FILE

     INPUT FILE: DSC2:CONTROL/OBJ

   REWIND INPUT FILE? Y
CNTROL    ?    C
END-OF-FILE

     INPUT FILE: DSC2:TXLIB/OBJ
   REWIND INPUT FILE? Y
TXROOT    ?    C
IMGLDR    ?    C
EVENTK    ?    S
TITTCM    ?    C
CRTPRO    ?    C
STA913    ?    C                    S if no 913 VDT
SVC913    ?    C                    S if no 913 VDT
STA911    ?    S                    C if 911 VDT
SVC911    ?    S                    C if 911 VDT
IOSUPR    ?    C
TSKFUN    ?    C
TSKLDR    ?    C
CNVRSN    ?    C
MEMSVC    ?    C
TBUFMG    ?    C
DTASK     ?    C
TXSTRT    ?    C
TXEND     ?    C
STASK     ?    C
END-OF-FILE

     INPUT FILE: *
END OBJECT MANAGER
```

8. Remove diskette #1 from drive #2 and load diskette #2 or #3 or #6 in drive #2.

9. Assemble TASKDF and TXDATA for linking with the selected TI-supplied modules as follows:

```
TXDS  936215  **  1/  0  0: 40

  PROGRAM:  :TXMIRA/SYS                      Assemble TXDATA
    INPUT:  :TXDATA/SRC
   OUTPUT:  :TXDATA/OBJ,:TXDATA/LST          Put listing on diskette
  OPTIONS:  CLS
TXMIRA 936227  **

TXDS 936215 ** 1/ 0 0: 46

  PROGRAM:  :TXMIRA/SYS                      Assemble TXDF
    INPUT:  :TASKDF/SRC
   OUTPUT:  :TASKDF/OBJ,:TASKDF/LST          Put listing on diskette
  OPTIONS:  CLS
TXMIRA 936227  **
```

10. Link the system as follows:

```
TXDS  936215  **  1/  0  0: 47

  PROGRAM:  :TXLINK/SYS
    INPUT:  :TXDATA/OBJ, :TASKDF/OBJ,:TXPART/OBJ
   OUTPUT:  :NEWSYS/SYS,:NEWSYS/LST
  OPTIONS:  CLITX990
TXLINK 937537  **
```

11. Remove diskette #2 or #3 or #6 from drive #2 and load diskette #1 in drive #2.

12. Mark the newly-linked system as the system file as follows:

```
TXDS  936215  **  1/  0  0: 54

PROGRAM:  :SYSUTL/SYS*

TX990 SYSTEMS UTILITY  937544  **

OP: .SF, :NEWSYS/SYS. TE.
  0:55:35      JAN  1,      0

TXDS 936215 ** 1/ 0 0: 55

PROGRAM:
```

13. Load into memory the new TX990 Operating System by pressing the HALT, RESET, and LOAD keys on the Programmer Panel:

```
TX990 SYSTEM                        RELEASE 2.2
MEMORY SIZE (WORDS): 24576  AVAILABLE: 14631
```

14. Bid TXDS via OCP by entering an exclamation point (!) to bid OCP; then respond to the period (.) prompt as follows:

```
!
. EX, 16. TE.                                    (Execute TXDS)
TXDS  936215  **  1/  0  0: 0
```

15. Remove diskette #1 from diskette drive #2 and insert diskette #2 into drive #2.

16. Copy TXDATA, TASKDF and LINK MAP LISTINGS to LP as follows:

```
  PROGRAM:  :TXCCAT/SYS
    INPUT:  :TXDATA/LST,:TASKDF/LST,:NEWSYS/LST
   OUTPUT:  LP
  OPTIONS:  RO
TXCCAT 937543  **
```

SECTION X

DISKETTE/DISC BACKUP AND INITIALIZE PROGRAM

10.1 INTRODUCTION

The Diskette Backup and Initialize Utility Program (BACKUP) may be used for two functions: initializing a diskette and copying (backing up) files to a diskette. It allows the user to: back up an entire diskette, partially backup a diskette, simply initialize a new diskette, or partially backup several diskettes to a single diskette.

The backup utility can copy and verify files, or copy or verify files separately. Errors are listed at the system console. Files that have no data are identified by warning messages.

Before writing to the output diskette, BACKUP checks to see if it is initialized. If not, it requests the user to initialize the diskette, prompting for required information. When copying files, BACKUP generates contiguous files on the output diskette, thus reducing fragmentation of disk space.

BACKUP also initialize the directory and allocation bit map on the output diskette, and can designate a system file if requested by the user.

10.2 LUNOs AND THEIR USES

LUNOs 5, 6, 10 and 11 are used. All LUNOs are assigned by BACKUP and are released when BACKUP terminates. LUNO 5 is assigned to the output diskette. LUNO 6 is assigned to the input diskette. LUNO 10 is assigned to the input file, and LUNO 11 is assigned to the output file.

10.3 OPERATING PROCEDURE

To load the Diskette Backup Utility and place it in execution under TX990, perform the following steps using the OCP:

1. Place the object module for BACKUP in the appropriate device and ready the device.

2. Enter the appropriate command to load the module. For example:

LP,CS2.	Loads the object code from cassette drive 2.
LP,DSC:BACKUP/SYS	Loads the object code from diskette file DSC:BACKUP/SYS.

3. Enter the appropriate command to execute the task.

EX,10.TE.	Execute the program just loaded. If the backup utility was linked to the system, the user need only to execute the task ID assigned to it at that time.

To place the backup utility under execution using the Terminal Executive Development System, perform step 1 above. If the TXDS Control Program is not executing, bid task 16_{16} if OCP is active.

EXECUTE,16.TE.

TXDS lists its heading and asks for input as follows:

```
TXDS   936215   **   011/77   2:10

PROGRAM:   DISC:BACKUP/SYS*
```

The user enters DSC:BACKUP/SYS* in response to the prompt: "PROGRAM:" The object code in the diskette file, DSC:BACKUP/SYS, is loaded and executed by the control program. Alternatively, the user can enter the cassette drive name.

```
PROGRAM:   CS1*
```

Then the object code on cassette drive 1 is loaded and executed.

10.4 USER INTERACTION WITH THE BACKUP UTILITY

When BACKUP is executed, it displays a heading:

```
BACKUP & INITIALIZE UTILITY PART NO. 936213*B
```

After the heading is displayed, BACKUP requests the output diskette name:

```
OUTPUT DISC OR VOLUME NAME?
```

The user enters either a device name or the volume name of the diskette to be initialized or copied to.

NOTE

A user response of "*" to any BACKUP prompts terminates the utility. A user response of "&" to any prompt returns BACKUP to the above prompt, "OUTPUT DISC OR VOLUME NAME?".

After the user specifies the output diskette, BACKUP checks it to see if it is initialized. If so, BACKUP prompts the following:

```
DELETE ALL FILES ON XXXX? (Y/N)
```

where XXXX is the output device name. The user enters Y if he desires to delete any preexisting files on the diskette. BACKUP clears all directory entries and resets the allocation bit map on the diskette. If the user enters N, BACKUP prompts the "INITIALIZE XXXX" message as described below.

If the output diskette is not initialized, BACKUP displays the following:

```
THE OUTPUT DISC MUST BE INITIALIZED
```

If the diskette must be initialized, or the user response to the "DELETE ALL FILES" message was N, BACKUP prompts the following:

```
INITIALIZE XXXX? (Y/N)
```

where XXXX is the output device name. If the user answers Y, the diskette is initialized. BACKUP prompts the message:

OUTPUT DISC ID

The user must enter a 1 to 32 character title for the diskette. The title is displayed by the SYSTUL Map Disc (MD) command. The following message is then displayed:

OUTPUT VOLUME NAME

The user may enter a 1 to 4 character volume name. The volume name will be used to access files on the diskette if volume name support is included in the user's customized TX990 Operating System. The volume name should not be the same as a device name. If the user enters only a carriage return, no volume name is defined for the diskette. Diskette initialization takes about four minutes.

In order to copy files to a diskette without destroying all of the files on the diskette, the user must respond N to both the "DELETE ALL FILES" and "INITIALIZE" prompts.

After the diskette is initialized, erased, or left intact, the user may copy and/or verify the files on any diskette. BACKUP prompts the following two messages, in order:

COPY FILES? (Y/N)
VERIFY FILES? (Y/N)

If the user responds with a Y to either prompt, BACKUP requests the pathname of the files to be copied/verified, as follows:

INPUT PATHNAME?

Valid responses must have one of the following formats.

VOL	ENTIRE DISKETTE
VOL:FILE/	ALL FILES ON THE DISKETTE OR VOLUME WITH THIS FILE NAME
:FILE/	ALL FILES ON DEFAULT DISKETTE WITH THIS FILE NAME
VOL/EXT	ALL FILES ON THE DISKETTE OR VOLUME WITH THIS EXTENSION
/EXT	ALL FILES ON DEFAULT DISKETTE WITH THIS EXTENSION
VOL:FILE/EXT	ONLY THIS FILE
VOL:FILE	ONLY THIS FILE
:FILE/EXT	THIS FILE FROM THE DEFAULT DISC
:FILE	THIS FILE FROM THE DEFAULT DISC

where VOL = DISKETTE NAME OR VOLUME NAME
FILE = FILE NAME
EXT = FILE NAME EXTENSION

If the output diskette already contains a file with the same name as one of the input files, the existing file is deleted and the copied file replaces it. After the specified files are copied and/or verified, BACKUP repeats the sequence with the "COPY FILES" prompt.

When the user responds N to both the "COPY FILES" and "VERIFY FILES" prompts, BACKUP prompts the user to designate a system file. A system file is the image of a TX990 system which may be booted (eg. :SYSASR/CMP). To insure that the file is on the output diskette, the diskette name is also prompted, as follows:

SYSTEM FILE PATHNAME? DSC2:

where DSC2 is the value given by the user in response to the "OUTPUT DISC" prompt.

The user may then enter the name of a file on the output diskette which is to be the system file. If only a carriage return is entered, no system file is designated.

After the "SYSTEM FILE" prompt, BACKUP returns to the "OUTPUT DISC OR VOLUME NAME" prompt. The utility may be terminated by a response of "*" to any prompt.

When BACKUP terminates, it displays the following message:

BACKUP & INITIALIZE UTILITY ENDED

10.5 ERROR MESSAGES AND RECOVERY

Error	Reason	Recovery
The OUTPUT DISC IS NOT USABLE. BACKUP UTILITY ABORTED.	Bad diskette	Get another diskette.
pathname DOES NOT VERIFY.	Bad copy	Rerun the backup utility.
pathname WAS EMPTY.	A file was created on the input file but was never used.	
DISC NAMES CANNOT BE THE SAME.	User entered same name for input and output disc names.	Reenter the disc names.
THE OUTPUT DISC MUST BE INITIALIZED.	Output diskette has never been initialized.	Execute the diskette initialization program.
pathname WAS UNSUC-CESSFULLY OPENED.	Disc was offline or diskette was full.	Ready the disc.

Error	Reason	Recovery
COULD NOT GET MEMORY. BACKUP UTILITY ABORTED.	There is not enough memory to run the backup utility.	
pathname HAD AN I/O ERROR, CODE nn.	nn is the error code; see error appendix I.	Correct problem. Rerun backup utility.
THE INPUT DISC HAS A BAD DIRECTORY RECORD. THE FILES IN THAT RECORD WERE NOT COPIED.	There is a diskette flaw in that record.	None.
THE OUTPUT DISC HAS A BAD DIRECTORY RECORD. THE OUTPUT DISC IS NOT USABLE.	There is a Diskette flaw on the output diskette.	Disregard the diskette.
DISC I/O ERROR CODE NN.	nn is the error code; see Appendix I – I/O Error Codes.	Correct error and retry.
FILE NOT FOUND.	The system file does not exist.	Select another file as the system file.

NOTE

The pathname parameter will be filled in with the actual pathname of the file.

SECTION XI

OBJECT MANAGER (OBJMGR) UTILITY PROGRAM

11.1 INTRODUCTION

The Object Manager (OBJMGR) Utility Program is used to collect object modules from several different diskette or cassette files and combine them into a single output file. The ability to do so is especially necessary when preparing to link all of the modules of a customized TX990 Operating System, since the TXDS Linking Utility can only accept up to three input files.

11.2 LUNOs

The OBJMGR Utility Program uses LUNO 0, LUNO 4, and LUNO 5. The system console, which is assigned to LUNO 0, is used by the OBJMGR Utility Program to request operator input information. LUNO 4 is used to read the input file, and LUNO 5 is used to write the output file. Both LUNO 4 and LUNO 5 are assigned and released by the OBJMGR Utility Program.

11.3 LOADING OBJMGR

OBJMGR executes using the TX990 Operating System in conjunction with the OCP module, or with the TXDS Control Program, or using the DX10, release 2.2 Operating System. The following paragraphs describe how to load and execute OBJMGR under each operating system.

11.3.1 LOADING OBJMGR USING THE TX990 OPERATING SYSTEM AND OCP. Load the program as follows:

1. After loading the OCP in accordance with the procedure itemized in the section in this manual entitled Operator Communication Package, observe the printout or display on the system console of the period (.) prompt.

2. Place the object module for OBJMGR in either the cassette or diskette drive and ready the device.

3. Using the LP (Load Program) command, load the OBJMGR object module from a cassette or diskette by using one of the following indicated methods:

   - LP,CS1,3 — Load OBJMGR at priority level 3 when the object module is on the cassette in cassette drive 1.
   - LP,DSC:OBJMGR/SYS. — Load OBJMGR from the diskette in diskette drive 1.

4. Enter the EX (Execute) command to execute OBJMGR and terminate OCP as follows:

   ```
   .EX,10.TE.
   ```

5. Observe the following printout or display on the system console when OBJMGR begins execution:

   ```
   990 OBJECT MANAGER 939870**
   ```

11.3.2 LOADING OBJMGR USING THE TX990 TERMINAL EXECUTIVE DEVELOPMENT SYSTEM. Load the program as follows:

1. Place the object module for OBJMGR in either the cassette or diskette drive and ready the device.

   If the TXDS control program is not active enter an "!" and observe the following display:

   ```
   TXDS   936215   *A   088/77   3:14
   PROGRAM:
   ```

2. When the OBJMGR object module is in cassette drive 1, enter "CS1*" in response to the prompt, as follows:

   ```
   PROGRAM CS1*
   ```

   When the OBJMGR object module is on a diskette file called :OBJMGR/SYS, enter the following:

   ```
   PROGRAM: :OBJMGR/SYS*
   ```

3. Observe the following printout or display on the system console when OBJMGR begins execution:

   ```
   990 OBJECT MANAGER 939870**
   ```

11.3.3 LOADING AND EXECUTING OBJMGR USING DX10, RELEASE 3.0. Perform the following steps on a DX10 release 3.0 system:

1. Place the OBJMGR object module in some sequential media which is readable by DX10. If the module is on cassette, place it in a cassette drive. If the module is on a diskette file, use the TXDX conversion utility available under DX10 to convert the object module file to a DX10 file.

2. Install the object module as a task, using the DX10 IT command:

   ```
   IT PF = .SPROGA, TN = OBJMGR, OBJ = <acnm>
   ```

   where <acnm> is the DX10 access name of the file or device which contains the object module.

3. Execute the object manager, using the DX10 XTS command.

   ```
   XTS   TN = OBJMGR
   ```

4. Upon execution, OBJMGR displays the following message:

   ```
   990 OBJECT MANAGER 945672*C
   ```

11.4 OPERATOR INTERACTION

After the initial heading is displayed, the object manager requests the following information:

OUTPUT FILE: <pathname of a sequential file or device>

INPUT FILE: <pathname of a sequential file or device>

REWIND INPUT FILE? { Y / N }

"IDT of an object module"? { C / S / I / R / A / D }

NOTE

If an "*" is entered in response to any prompt, OBJMGR writes an end-of-file to the output file, closes the file, releases all LUNOs and terminates. If an "&" is entered, OBJMGR performs all of the above except terminate; it then restarts with the "OUTPUT FILE" prompt.

The first prompt made by OBJMGR is the following:

OUTPUT FILE:

The user must enter a complete pathname for a sequential device or file. There are no defaults. DX10 users must not use synonyms. The object modules selected by the user in later steps are written to this pathname.

The next prompt is the following:

INPUT FILE:

The user must enter the name of a sequential file or device which contains object modules to be edited onto the output file. In addition to modules on this input file, the user may insert new modules from other files.

After the input file pathname is entered, OBJMGR prompts the following:

REWIND INPUT FILE?

The user must answer either Y (yes, rewind the input file) or N (do not rewind the file).

After the input file is ready, OBJMGR reads the first record, and displays the IDT (see the Assembly Language Programmer's Guide) for that module, followed by a question mark. The user may then choose to include or exclude that module from the output file, or to insert modules from other files, by entering one of the following action commands:

C – copy the object module
S – skip the object module
I – insert object modules from other files

R – replace this module with modules from other files (combination of S and I)
A – copy all remaining object modules in the input file
D – done with this input file; leaves the file positioned at the current IDT for future use.

The Copy (C) command causes the object manager to copy the module with the displayed IDT to the output file.

The Skip (S) command causes OBJMGR to skip the module with the displayed IDT without copying it.

The Insert (I) command allows the user to insert modules from another file (the insert file) onto the output file, before the module with the displayed IDT. After the I command is given, OBJMGR prompts the "INPUT FILE" and "REWIND" messages. The user enters the new file, and Y or N, and OBJMGR reads the first record of the new input file and displays the IDT of the first object module. The user may enter any of the action commands except I or R. After the user enters a Done (D) command, OBJMGR skips several lines and displays a line of asterisks, followed by the IDT that was displayed when the Insert command was entered. If an end-of-file on the insert file is read, an "END-OF-FILE" message is displayed, followed by the IDT that was displayed before the Insert command. The user may then enter any command.

The Replace (R) command causes OBJMGR to skip the object module with the displayed IDT, and then insert modules from another file (the replace file). After skipping the original module, the Replace command executes exactly like the Insert command.

The All (A) command causes the object manager to copy all of the remaining object modules in the input file to the output file, and then resume execution at the "INPUT FILE" prompt.

The Done (D) command causes OBJMGR to backspace the current file one record (so that the IDT which was displayed will be displayed next time the file is opened without rewind). If D is issued while processing a Replace or Insert command, OBJMGR resumes prompting of IDTs on the original input file. If D is issued for the input file, execution is resumed by prompting for a new "INPUT FILE".

11.5 ERROR MESSAGES

Table 11-1 shows the error messages which may be returned by OBJMGR.

Table 11-1. Error Messages

Error Message	Description	Recovery
CAN'T ASSIGN I/O	Attempt to assign LUNO for input or output was unsuccessful.	Check device name, and enter correct name. If system capacity for active LUNOs has been reached, release a LUNO that is no longer required. The user may terminate Object Manager and attempt to assign the LUNO with an ALUNO command to obtain a more specific error message.
INPUT ERROR	I/O error during keyboard entry.	Try again. If error persists, check input device and interface.
CAN'T OPEN I/O, TERMINATING	Error during open operation for input or output.	Object Manager terminates. Obtain correct object module or add proper first record and restart Object Manager.
INVALID IDT RECORD	First record of object module does not contain identifier.	Object Manager terminates. Obtain correct object module or add proper first record and restart Object Manager.
INPUT ERROR, RETRY?	Error while reading input file.	Enter N to use the record as read. Enter Y to backspace and reread the record. When record is a card, user must remove card from output stacker and place it in input hopper. Enter a carriage return to terminate.
OUTPUT FILE ERROR, TERMINATING	Error while writing output file.	Object Manager terminates. Correct the problem and restart Object Manager.
EOF EMBEDDED IN OBJECT MODULE	An end-of-file record has been detected in object module.	Object Manager terminates. Obtain correct object module or remove misplaced record and restart Object Manager.
INVALID COMMAND C = COPY, S = SKIP, R = REPLACE, I = INSERT, D = DONE, A = ALL	Invalid response to request.	Enter correct response. An asterisk terminates OBJMGR, and an ampersand (&) restarts it.

SECTION XII

LIST80/80 (LIST80) UTILITY PROGRAM

12.1 LIST80/80

The LIST80/80 utility program copies 80-character records from one device or file to another. Optionally, LIST80/80 adds carriage control characters for displaying or printing. All files must previously have been created and all files and devices must previously have been rewound. LIST80/80 can be run from OCP control only and cannot be run using the TXDS Control Program.

12.1.1 LOAD AND EXECUTING LIST80/80. To execute LIST80/80, place the object code in the appropriate device and perform the following steps:

NOTE

When LIST80/80 is linked into the Operating System during system generation, step 1 does not apply.

1. Enter a command to load LIST80/80 in the dynamic task area. The following are examples of commands to install LIST80/80:

.LP,CS1,3.	Load task LIST80/80 at priority level 3 when the object module is on cassette unit 1.
.LPROG,CR,3.	Load task LIST80/80 at priority level 3 when the object module is in the card reader.
.LP,:LIST80/SYS,3.	Load task LIST80/80 at priority level 3 when the object module is in the disc file DCS:LIST80/SYS.

2. Assign LUNO 10_{16} to the input device. The following example applies when the input device is Cassette Unit 2. (The cassette must have been written in the line mode with records that contain 80 or fewer characters.)

   .ALUNO,10,CS2.

3. Assign LUNO 11_{16} to the output file. The following example applies when the output file is DSC2:SOURCE/SRC.

   .ALUNO,11,DSC2:SOURCE/SRC.

4. Enter the following command to execute LIST80/80 and terminate OCP.

   .EXECUTE,10.TE.

NOTE

When LIST80/80 is linked into the system during system generation, the task identifier assigned during system generation is used instead of 10 in the command in step 4.

5. LIST80/80 displays a title and part number heading, as shown:

   LIST80/80 937976*A

6. LIST80/80 reads the records from the input device, adds carriage control characters, and writes the records to the output device.

7. When LIST80/80 finishes, a message is displayed on the LOG.

   LIST80/80 TERMINATED

When the output device is a cassette unit, LIST80/80 does not add carriage control characters. An alternative command may be entered in step 4, as follows:

.EXECUTE,10,FFFF.TE.

When the Execute command is entered in this form (or with any nonzero value instead of FFFF), LIST80/80 does not add carriage control characters to the output record, regardless of the output device.

12.1.2 **LIST80/80 ERROR MESSAGES.** LIST80/80 prints two error messages. One error message prints as follows:

ERROR DETECTED ON OPEN OF LUNO 10 OR 11

The possible reasons for this error are:

- LUNO 10 assigned to an inappropriate device.
- LUNO 11 assigned to an inappropriate device.
- An error other than busy was returned by the OPEN supervisor call.

The following error message is printed when an unsuccessful read operation occurs on LUNO 10:

READ ERROR ON LUNO 10.

The following error message is printed when an unsuccessful write operation occurs on LUNO 10:

WRITE ERROR ON LUNO 11.

SECTION XIII

DISKETTE DUMP (DSKDMP) UTILITY PROGRAM

13.1 INTRODUCTION

The Diskette Dump Utility (DSKDMP) Program allows users to load, dump and modify FD800 Floppy Disc diskettes on an allocation unit (AU) basis. (Refer to Section IV of this manual for a description of the allocation unit and its application to file management.) The utility can be executed from either OCP or TXDS.

In addition to the diskette, a V911 or V913 Video Display Terminal is required.

13.2 LUNOs

Diskette Dump uses Video Display Terminal (VDT) station number 1, LUNO 4 and LUNO 1. It uses VDT station number 1 to display sectors to the user and accept commands from the user. It assigns LUNO 4 internally to the floppy-disc drive whose diskette is being dumped, loaded, or modified.

The utility uses LUNO 1 when the print directive (P) is entered. Therefore, the user must assign LUNO 1 before the print directive is exercised.

13.3 LOADING PROCEDURES

To load the Diskette Dump Utility and execute it using OCP, perform the following steps by entering commands at the keyboard of the system console.

1. Place the object module for Diskette Dump in the appropriate device and ready the device.

2. Enter a command to install Diskette Dump in the dynamic task area. The following are examples of load program (LP) commands to install Diskette Dump:

   .LP,CS1. Load Diskette Dump at priority level 3 when the object module is Cassette Unit 1.

   .LP,CR. Load Diskette Dump at priority level 3 when the object module is in the Card Reader.

3. Assign LUNO 1 to a printing device if the print directive is to be exercised. The following is a sample command:

   .AL,1,LP. Assign LUNO 1 to the line printer.

4. Enter a command to execute Diskette Dump and terminate OCP, as follows:

   .EXECUTE,10.TE.

If Diskette Dump was linked into the system at system generation, the load program command is not entered and the operand of the execute command (EXECUTE) is the task identifier assigned to Diskette Dump when the system was generated.

To initiate execution of the Diskette Dump utility using the Terminal Executive Development System perform step 1 above, followed by the steps below:

1. Bid TXDS control program when OCP is not active by entering "!". Otherwise, enter "!" to bid OCP and enter "EX,16.TE." to bid TXDS control program.

2. The TXDS control program displays a heading and prompts for input as follows:

```
TXDS 936215 *A 011/77 2:10
PROGRAM:
```

3. Enter :DSKDMP/SYS* in response to the prompt as follows:

```
PROGRAM: :DSKDMP/SYS*
```

13.4 OPERATING PROCEDURES

Upon execution, an initial mask is displayed on the station number 1 VDT with the cursor positioned to the first field of the command line.

The following shows the initial VDT mask:

```
                990 DISKETTE DUMP/LOAD              937562**
00-0F
10-1F
20-2F
30-3F
40-4F
50-5F
60-6F
70-7F
    DISC: ____      AU NO:__      SECTOR:_
```

The cursor is positioned under the first space after DISC:. The user now must enter, in the proper field of the command line, the disc drive name, AU number and sector number which is to be displayed on the VDT. To advance from one field of the command line to the next, the user either enters the maximum number of characters allowed in the field or presses the TAB key. To return to a previous field, the user presses the NEW LINE key. For example, if the cursor was positioned at the SECTOR: field and the user wished to return to the DISC: field, the user would press the NEW LINE key twice.

Each time data is entered in a field, the utility validates the data. If the data is not correct, the message WRONG! is displayed in the lower-right-hand corner of the VDT screen. Before the user may advance from one field to the next, valid data must be entered in the field from which the user wishes to advance. The following is a list of restrictions on the values for each of the command line fields:

DISC: Requires a one- to four-character disc-drive name. This name is determined at system generation time.

AU NO: Requires a one- to three-digit decimal number. The range of AU numbers is 0 to 332. Leading zeros are not required. Hitting the TAB key without entering a number causes a zero to be entered.

SECTOR: Requires a one-digit decimal number. The range of the sector number is 0 to 5. Hitting the TAB key without entering a number causes zero to be entered.

After the user has entered the disc-drive name, AU-number, and sector-number parameters, the utility displays the sector on the VDT, indicates the mode of the data being displayed (either ASCII, EBCDIC, or hexadecimal and positions the cursor, as indicated by the blinking light, to accept utility directives. The following is an example of the VDT display:

```
                    990 DISKETTE DUMP/LOAD          937562  ◆◆

DISC: DSC_   A.U. :   5    SECTOR:   0    MODE: ASCII

00--0F          F  M    P  L    I  B       0    B  J   00 06    W 00    00 00
10--1F          D  S    R  L    I  B       0    B  J   00  ◆    W 00    00 00
20--2F          0  C    P  L    I  B       0    B  J   00  I    W 00    00 00
30--3F          T  X    L  I    B          0    B  J   00 5D    W 00    00 00
40--4F          6  E    N  T    X          3    Y  3   00 30    W 00    00 00
50--5F          .  .    .  .    .  .    .  .    .  .    .  .    .  .    .  .
60--6F          .  .    .  .    .  .    .  .    .  .    .  .    .  .    .  .
70--7F          .  .    .  .    .  .    .  .    .  .    .  .    .  .    .  .
```

The user can now enter one of the Blinking Cursor one-character directives indicated in table 13-1. The following paragraphs define these functions.

13.4.1 INCREMENT SECTOR NUMBER (I). This directive increments the number of the sector being displayed and causes the new sector to be displayed. If the new sector number goes beyond 5, the first sector (sector 0) of the next AU is displayed.

13.4.2 DECREMENT SECTOR NUMBER (D). This directive decrements the number of the sector being displayed and causes the new sector to be displayed. If the new sector number goes lower than 0, the last sector (sector 5) of the previous AU is displayed.

13.4.3 PRINT DISPLAY (P). This directive causes the displayed sector to be written to the hard-copy device assigned to LUNO 1. After P is input, the following message is displayed:

PRINT? (Y/N)

The user can then assign LUNO 1 to the hard-copy device, if it has not already been done. The following is an example of the command that may be entered on the system console.

.AL,1,LOG.TE.

After assigning LUNO 1, the user types a Y on the VDT and the output is performed. Typing an N indicates to the utility that the user decided not to print the display and so the printing process is not started.

13.4.4 SET DATA MODE TO ASCII (A). This directive causes the sector data to be interpreted and displayed in ASCII format.

13.4.5 SET DATA MODE TO EBCDIC (E). This directive causes the sector data to be interpreted and displayed in EBCDIC format.

Table 13-1. Diskette Dump Utility Directives

Directive	Function
I	Increment sector number
D	Decrement sector number
P	Print display
A	Set data mode to ASCII
E	Set data mode to EBCDIC
H	Set data mode to Hexadecimal
M	Modify displayed sector data
New-Line-Position cursor to SECTOR: field.	

13.4.6 SET DATA MODE TO HEXADECIMAL (H). This directive causes the sector data to be displayed in hexadecimal format.

Figure 12-1 shows the same sector displayed in the three different modes: ASCII, EBCDIC and hexadecimal. The displayed sector is grouped into 8-bit (byte) fields. Therefore on each line of the display there are 16 bytes of the sector displayed. On the hexadecimal (hex) display each byte is represented by two hexadecimal characters. On the ASCII and EBCDIC displays, each byte is represented as either a two-character upper-case ASCII or EBCDIC character (first character is blank), a two-character lower-case ASCII or EBCDIC character (first character is a "."), or if the byte of data has no ASCII or EBCDIC representation, the two-character hexadecimal value is displayed.

13.4.7 MODIFY DISPLAYED SECTOR DATA (M). This directive allows a user to modify the displayed sector data. After the user enters the directive, the following message is output:

MODIFY? (Y/N):

If the user types N, the utility stops the modify operation. If the user types Y, the cursor positions to the first byte displayed for the sector (upper-left-hand corner of the display). The user may move the cursor about with the "arrow" keys up, down, right and left. Hexadecimal characters may be entered by entering the two character hex value. An ASCII character may be entered by typing a blank followed by the desired ASCII character. An EBCDIC character may be entered by typing a period followed by the desired EBCDIC character. After all changes have been made, the "new line" character must be entered. This causes the following message to be output:

RE-WRITE DISC?:

If the user types N, the modify operation is terminated without modifying the data on the disc. If the user types Y, the modified sector is written to disc.

13.4.8 POSITION CURSOR TO SECTOR: FIELD (NEW LINE). Typing New Line causes the cursor to position to the SECTOR: parameter field. Typing this character three more times causes Diskette Dump to terminate. It outputs the following message on the VDT screen.

**** END OF PROGRAM ****

13.5 ERROR MESSAGES

The following error messages can be generated during execution of DSKDMP:

WRONG!	An attempt was made to input invalid parameters, command directives, or to perform an invalid operation or LUNOs were not available to the task.
DISC ERROR	Either a disc hardware error occurred or an attempt was made to read a nonexistent AU.

```
                    990 DISKETTE DUMP/LOAD          937562  ◆◆

DISC: DSC2   A.U. :   5    SECTOR:   0    MODE: HEX

00-0F       46 4D   50 4C   49 42   20 4F   42 4A   00 06   57 00   00 00
10-1F       44 53   52 4C   49 42   20 4F   42 4A   00 2A   57 00   00 00
20-2F       4F 43   50 4C   49 42   20 4F   42 4A   00 49   57 00   00 00
30-3F       54 58   4C 49   42 20   20 4F   42 4A   00 5D   57 00   00 00
40-4F       47 45   4E 54   58 20   20 53   59 53   00 80   57 00   00 00
50-5F       2E 2E   2E 2E   2E 2E   2E 2E   2E 2E   2E 2E   2E 2E   2E 2E
60-6F       2E 2E   2E 2E   2E 2E   2E 2E   2E 2E   2E 2E   2E 2E   2E 2E
70-7F       2E 2E   2E 2E   2E 2E   2E 2E   2E 2E   2E 2E   2E 2E   2E 2E

DISC: DSC2   A.U. :   5    SECTOR:   0    MODE: ASCII

00-0F        F  M    P  L    I  B       □    B  J   00 06    W 00   00 00
10-1F        D  S    R  L    I  B       □    B  J   00  ◆    W 00   00 00
20-2F        □  C    P  L    I  B       □    B  J   00  I    W 00   00 00
30-3F        T  X    L  I    B          □    B  J   00 5D    W 00   00 00
40-4F        G  E    N  T    X          S    Y  S   00 80    W 00   00 00
50-5F        .  .    .  .    .  .    .  .    .  .    .  .    .  .    .  .
60-6F        .  .    .  .    .  .    .  .    .  .    .  .    .  .    .  .
70-7F        .  .    .  .    .  .    .  .    .  .    .  .    .  .    .  .

DISC: DSC2   A.U. :   5    SECTOR:   0    MODE: EBCDIC

00-0F       46  (    %  <    49 42   20 4F   42 4A   00 06   57 00   00 00
10-1F       44 53   52  <    49 42   20 4F   42 4A   00 2A   57 00   00 00
20-2F       4F 43    %  <    49 42   20 4F   42 4A   00 49   57 00   00 00
30-3F       54 58    <  49   42 20   20 4F   42 4A   00  )   57 00   00 00
40-4F       47 45    +  54   58 20   20 53   59 53   00 80   57 00   00 00
50-5F       2E 2E   2E 2E   2E 2E   2E 2E   2E 2E   2E 2E   2E 2E   2E 2E
60-6F       2E 2E   2E 2E   2E 2E   2E 2E   2E 2E   2E 2E   2E 2E   2E 2E
70-7F       2E 2E   2E 2E   2E 2E   2E 2E   2E 2E   2E 2E   2E 2E   2E 2E

(A)137506
```

Figure 13-1. Three Modes of Sector Display

APPENDIX A

LIST OF SUPERVISOR CALLS FOR USER TASKS AND FILE MANAGEMENT OF I/O DEVICE-FILES

APPENDIX A

LIST OF SUPERVISOR CALLS FOR USER TASKS AND FILE MANAGEMENT OF I/O DEVICE-FILES

Table A-1 lists the supervisor calls available to the user. The call code, the hexadecimal value to be placed in byte 0 of the supervisor call block (SCB), is shown in the first column. The name of the supervisor call is listed in the second column, followed by the byte-size of the supervisor call block in the third column. The fourth column contains an X if the (SCB) must be word-aligned. The fifth column contains a brief description of the results of the supervisor call. For a detailed description of each supervisor call, refer to Sections III and IV herein this manual. It should be noted that "R" signifies "Workspace Register" in table A-1 below.

Table A-1. Supervisor Calls

Code	Supervisor Call	Byte Size	Aligned	Result
00_{16}	File Management	Note 1	X	Note 1
01_{16}	Wait for I/O	4	X	Calling task suspended pending completion of I/O operation defined in SCB at the address in bytes 2 and 3 of SCB.
02_{16}	Time Delay	4	X	Calling task suspended for number of system time units in bytes 2 and 3 of SCB.
03_{16}	Date and Time	4	X	Date and time returned in binary form in five-word area at address in bytes 2 and 3 of SCB.
04_{16}	End of Task	1		Calling task terminated.
05_{16}	Bid Task	Note 2	X	Task (ID in byte 2 of SCB) is bid.
06_{16}	Unconditional Wait	1		Calling task suspended unconditionally.
07_{16}	Activate Suspended Task	3		Suspended task (ID in byte 2 of SCB) is activated.
08_{16}	Character Input	3		Input character from station keyboard (ID in byte 2 of SCB); place in R0.
09_{16}	Do Not Suspend	2		Inhibit suspension of calling task by system for number of system time units determined by contents of byte 1 of SCB.
$0A_{16}$	Convert Binary to Decimal	8		Convert binary value in R0 to decimal ASCII characters; place in bytes 2 through 7 of SCB.

Table A-1. Supervisor Calls (Continued)

Code	Supervisor Call	Byte Size	Aligned	Result
$0B_{16}$	Convert Decimal to Binary	8		Convert decimal ASCII characters in bytes 2 through 7 of SCB to binary value; place in R0.
$0C_{16}$	Convert Binary to Hexadecimal	6		Convert binary value in R0 to hexadecimal ASCII characters; place in bytes 2 through 5 of SCB.
$0D_{16}$	Convert Hexadecimal to Binary	6		Convert hexadecimal ASCII characters 2 through 5 of SCB to binary value place in R0.
$0E_{16}$	Activate Time Delay Task	3		Time delay task (ID in byte 2 of SCB) is activated.
$0F_{16}$	Abort I/O	2		I/O in progress with LUNO in byte 1 is terminated.
10_{16}	Get COMMON Data Address	2		Address of COMMON area in R9; size in R8.
11_{16}	Change Priority	2		Priority of calling task changed to value in byte 1 of SCB.
12_{16}	Get Memory	4	X	The number of 32-byte blocks of memory requested in bytes 2 and 3 of the SCB is allocated and the address of the first block is placed in R9.
13_{16}	Release Memory	4	X	Included for compatibility with DX10; performs no operation.
15_{16}	File Management	12	X	The file management operation defined in the SCB (Assign or Release LUNO) is performed.
16_{16}	End of Program	1		Operation bids the rebid task which is usually the TXDS Control Program.
17_{16}	Get Parameters	6	X	Task parameters are placed in bytes 2 through 5 of SCB.
18_{16}	Conditional Character Input	3		Bit 2 of Status Register set to one and character from station keyboard (ID in byte 2 of SCB) placed in R0, or bit 2 set to zero if no character available.
$1A_{16}$	VDT Utility	12	X	The VDT I/O operations defined in the SCB are performed. (Write or read characters, open or close I/O, position cursor or tab.)

Table A-1. Supervisor Calls (Continued)

Code	Supervisor Call	Byte Size	Aligned	Result
$1B_{16}$	Return COMMON	1		Included for compatibility with DX10; performs no operation.
$1C_{16}$	Put Data	14	X	Put message on message queue.
$1D_{16}$	Get Data	14	X	Get message from message queue.
$1E_{16}$	Abort I/O SVC	4	X	Abort I/O on specific SVC.
20_{16}	Get Own ID	2		Task ID returned in byte 1.
21_{16}	Get System Table	4	X	System table address returned in bytes 2 and 3.
23_{16}	Make Task Privileged	2		Makes the task privileged.

Notes:

1. Size of SCB for File Management supervisor call includes 2-byte SCB and 10- or 14- or 22-byte SCB. I/O operation defined in SCB is performed.

2. Size of SCB for Bid Task supervisor call is 3, 6, or 8 bytes. When two optional parameter words are included, 8 bytes are required; when a single optional parameter word is included, 6 bytes are required; otherwise, 3 bytes are required.

APPENDIX B

DEVICE CHARACTER SETS

APPENDIX B

DEVICE CHARACTER SETS

The TX990 Operating System supports ASCII I/O operations with all devices. This section lists the character sets for the devices and the end-of-record and end-of-file sequences for the devices.

Table B-1. 911 Video Display Terminal Character Set

End-of-Record - NEW LINE (CR)

Character	Hexadecimal	Character	Hexadecimal	Character	Hexadecimal
Space	20	1	31	B	42
!	21	2	32	C	43
"	22	3	33	D	44
#	23	4	34	E	45
$	24	5	35	F	46
%	25	6	36	G	47
&	26	7	37	H	48
'	27	8	38	I	49
(	28	9	39	J	4A
)	29	:	3A	K	4B
*	2A	;	3B	L	4C
+	2B	<	3C	M	4D
,	2C	=	3D	N	4E
-	2D	>	3E	O	4F
.	2E	?	3F	P	50
/	2F	@	40	Q	51
0	30	A	41	R	52

Table B-1. 911 Video Display Terminal Character Set (Continued)

Character	Hexadecimal	Character	Hexadecimal	Character	Hexadecimal
S	53	a	61	n	6E
T	54	b	62	o	6F
U	55	c	63	p	70
V	56	d	64	q	71
W	57	e	65	r	72
X	58	f	66	s	73
Y	59	g	67	t	74
Z	5A	h	68	u	75
[	5B	i	69	v	76
\	5C	j	6A	w	77
]	5D	k	6B	x	78
^	5E	l	6C	y	79
–	5F	m	6D	z	7A

Notes:

1. Entering left arrow (←) backspaces the cursor and deletes the character in the buffer.
2. Entering right arrow (→) moves the cursor one column to the right and places the character in the buffer.
3. Entering ERASE FIELD moves the cursor to the first character of the line, clears the characters on the line, and deletes those characters in the buffer.
4. Entering RETURN positions the cursor at the first character of the next line and terminates the record.

Table B-1. 911 Video Display Terminal Character Set (Continued)

Notes (Continued):

5. Table B-1 applies to the record and file modes only.
6. The maximum buffer size for record or file mode VDT is 82 characters.
7. Pressing the space bar stops output in record mode. Pressing the space bar again restarts output.
8. Pressing the ESC key terminates output.
9. Entering a tab stores 09_{16} in the buffer and writes a space to the screen.

Table B-2. 913 Video Display Terminal Character Set

End-of-Record - NEW LINE (C/R)

Character	Hexadecimal	Character	Hexadecimal	Character	Hexadecimal
Space	20	6	36	L	4C
!	21	7	37	M	4D
"	22	8	38	N	4E
#	23	9	39	O	4F
$	24	:	3A	P	50
%	25	;	3B	Q	51
&	26	<	3C	R	52
'	27	=	3D	S	53
(	28	>	3E	T	54
)	29	?	3F	U	55
*	2A	@	40	V	56
+	2B	A	41	W	57
,	2C	B	42	X	58
-	2D	C	43	Y	59
.	2E	D	44	Z	5A
/	2F	E	45	[	5B
0	30	F	46	\	5C
1	31	G	47	]	5D
2	32	H	48	^	5E
3	33	I	49	_	5F
4	34	J	4A		
5	35	K	4B		

Table B-2. 913 Video Display Terminal Character Set (Continued)

Notes:

1. Pressing the backspace arrow (←) moves the cursor one column to the left and deletes the character in the buffer.
2. Pressing the forward space arrow (→) moves the cursor one column to the right and places the character in the buffer.
3. Entering DELETE LINE moves the cursor to the first character of the line, clears the characters on the line, and deletes those characters in the buffer.
4. Entering NEW LINE positions the cursor at the first character of the next line and terminates the record.
5. Entering TAB causes 09_{16} to be stored in the buffer and a space to be written on the screen.
6. Table B-2 applies to the record and file modes only.
7. The maximum buffer size for record or file mode VDT is 82 characters.
8. Pressing the space bar stops output in record mode. Pressing the space bar again restarts output.
9. Pressing the RESET key terminates output.

Table B-3. 733 or 743 Data Terminal Character Set

End-of-Record - C/R (applies to keyboard and cassette input only)

End-of-file - DC3 (applies to keyboard and cassettes only)

Character	Hexadecimal	Character	Hexadecimal	Character	Hexadecimal
Space	20	7	37	N	4E
!	21	8	38	O	4F
"	22	9	39	P	50
#	23	:	3A	Q	51
$	24	;	3B	R	52
%	25	<	3C	S	53
&	26	=	3D	T	54
'	27	>	3E	U	55
(	28	?	3F	V	56
)	29	@	40	W	57
*	2A	A	41	X	58
+	2B	B	42	Y	59
,	2C	C	43	Z	5A
-	2D	D	44	[	5B
.	2E	E	45	\	5C
/	2F	F	46	]	5D
0	30	G	47	^	5E
1	31	H	48	_	5F
2	32	I	49	`	60
3	33	J	4A	a	61
4	34	K	4B	b	62
5	35	L	4C	c	63
6	36	M	4D	d	64

Table B-3. 733 or 743 Data Terminal Character Set (Continued)

Character	Hexadecimal	Character	Hexadecimal	Character	Hexadecimal
e	65	n	6E	w	77
f	66	o	6F	x	78
g	67	p	70	y	79
h	68	q	71	z	7A
i	69	r	72	{	7B
j	6A	s	73	¦	7C
k	6B	t	74	}	7D
l	6C	u	75	~	7E
m	6D	v	76		

Notes:

Keyboard:

1. BS character (08_{16}) is returned to printer as LF and BS. Deletes most recently entered character in buffer.
2. HT character (09_{16}) is returned to printer as space. Character is placed in buffer.
3. LF character ($0A_{16}$) is returned to printer as LF, but is not stored in buffer.
4. CR character ($0D_{16}$) is returned to printer as CR, and character is not placed in buffer. Character terminates the record.
5. DC3 character (13_{16}) is not stored in buffer. When DC3 is first character of record, the record is an end-of-file record.
6. ESC character ($1B_{16}$), when entered during output, terminates output with a write error.
7. DEL character ($7F_{16}$) is returned to printer as a line feed and carriage return. Character deletes current input record.
8. Maximum buffer size is 83 characters.

Printer:

1. BS character (08_{16}) results in a backspace operation.
2. HT character (09_{16}) results in printing a space.
3. LF character ($0A_{16}$) results in a line feed operation.
4. FF character ($0C_{16}$) results in eight line feed operations.
5. CR character ($0D_{16}$) results in a carriage return operation.
6. End-of-record occurs when specified number of characters have been printed.
7. Maximum buffer size is 83 characters.

Cassette Input:

1. The characters LF ($0A_{16}$) and DEL ($7F_{16}$) or the character DEL at the beginning of a record are ignored. The first valid character of the record is the character following the DEL.
2. The characters HT (09_{16}), FF ($0C_{16}$), BEL (07_{16}), and BS (08_{16}) are stored in the user's buffer.
3. The character ETB (17_{16}) is stored in the user's buffer as a CR ($0D_{16}$).

Table B-3. 733 or 743 Data Terminal Character Set (Continued)

Notes: (Continued)

4. The character DC3 (13_{16}) as the first character of a record indicates an end-of-file record. The system returns end of file status after positioning the tape at the beginning of the next record. The DC3 is not stored in the buffer.
5. The character CR ($0D_{16}$) indicates end-of-record, and is not stored in the buffer.
6. Maximum buffer size is 83 characters.

Cassette Output:

1. The end of block character sequence is CR ($0D_{16}$) LF ($0A_{16}$) DC4 (14_{16}) DEL ($7F_{16}$). The end of file character sequence is DC3 (13_{16}) CR DC4 DEL. These characters are supplied by the system, not by the user.
2. The characters HT (09_{16}), FF ($0C_{16}$), BEL (07_{16}), and BS (08_{16}) are written unchanged.
3. The character CR ($0D_{16}$) is translated to ETB (17_{16}) and written.
4. The character DC3 (13_{16}) may be placed within a record, but may not be the first character of a record other than the end of file record.
5. End-of-record occurs when specified number of characters have been written.
6. Maximum buffer size is 83 characters.

Table B-4. Card Reader Character Set

End-of-File /* (Cols. 1 and 2)

Character	Hexadecimal	Row Punches	Character	Hexadecimal	Row Punches
Space	20	None	8	38	8
!	21	11-8-2	9	39	9
"	22	8-7	:	3A	8-2
#	23	8-3	;	3B	11-8-6
$	24	11-8-3	<	3C	12-8-4
%	25	0-8-4	=	3D	8-6
&	26	12	>	3E	0-8-6
'	27	8-5	?	3F	0-8-7
(	28	12-8-5	@	40	8-4
)	29	11-8-5	A	41	12-1
*	2A	11-8-4	B	42	12-2
+	2B	12-8-6	C	43	12-3
,	2C	0-8-3	D	44	12-4
-	2D	11	E	45	12-5
.	2E	12-8-3	F	46	12-6
/	2F	0-1	G	47	12-7
0	30	0	H	48	12-8
1	31	1	I	49	12-9
2	32	2	J	4A	11-1
3	33	3	K	4B	11-2
4	34	4	L	4C	11-3
5	35	5	M	4D	11-4
6	36	6	N	4E	11-5
7	37	7	O	4F	11-6

Table B-4. Card Reader Character Set (Continued)

End-of-File /* (Cols. 1 and 2)

Character	Hexadecimal	Row Punches	Character	Hexadecimal	Row Punches
P	50	11-7	X	58	0-7
Q	51	11-8	Y	59	0-8
R	52	11-9	Z	5A	0-9
S	53	0-2	[	5B	12-8-2
T	54	0-3	\	5C	0-8-2
U	55	0-4	]	5D	12-8-7
V	56	0-5	¬	5E	11-8-7
W	57	0-6	–	5F	0-8-5

Notes:

1. End of record occurs when the specified number of characters, or 80 characters have been read.
2. Maximum buffer size is 80 characters.

Table B-5. Line Printer Character Set

Character	Hexadecimal	Character	Hexadecimal	Character	Hexadecimal
Space	20	9	39	R	52
!	21	:	3A	S	53
"	22	;	3B	T	54
#	23	<	3C	U	55
$	24	=	3D	V	56
%	25	>	3E	W	57
&	26	?	3F	X	58
'	27	@	40	Y	59
(	28	A	41	Z	5A
)	29	B	42	[	5B
*	2A	C	43	\	5C
+	2B	D	44	]	5D
,	2C	E	45	^	5E
-	2D	F	46	_	5F
.	2E	G	47	`	60
/	2F	H	48	a	61
0	30	I	49	b	62
1	31	J	4A	c	63
2	32	K	4B	d	64
3	33	L	4C	e	65
4	34	M	4D	f	66
5	35	N	4E	g	67
6	36	O	4F	h	68
7	37	P	50	i	69
8	38	Q	51	j	6A

Table B-5. Line Printer Character Set (Continued)

Character	Hexadecimal	Character	Hexadecimal	Character	Hexadecimal
k	6B	r	72	y	79
l	6C	s	73	z	7A
m	6D	t	74	{	7B
n	6E	u	75	¦	7C
o	6F	v	76	}	7D
p	70	w	77	~	7E
q	71	x	78	DEL	7F

Notes:

1. Models 306 and 588 printers respond to the following control characters:

   BS (08_{16}) results in a backspace operation.

   HT (09_{16}) results in a single space.

   LF ($0A_{16}$) results in a line feed operation.

   CR ($0D_{16}$) results in a carriage return operation.

   FF ($0C_{16}$) results in a form feed operation.

   BEL (07_{16}) results in a tone signal.

   The character SO ($0E_{16}$) results in character elongation for the line of characters following the SO. Elongation doubles the width of the characters, and a line of elongated characters contains one-half the number of characters on a normal line; i.e., 40 or 66.

2. Models 2230 and 2260 printers respond to the following control characters:

   HT (09_{16}) results in a single space.
   LF ($0A_{16}$) results in a line feed operation.
   CR ($0D_{16}$) results in a carriage return operation.
   FF ($0C_{16}$) results in a form feed operation.

   Only characters 20_{16} through $5F_{16}$ are supported. All other graphics character codes are converted to blanks. The codes $5E_{16}$ and $5F_{16}$ produce the characters ↑ and ←, respectively.

3. The Model 810 printer responds to the various control characters and control character sequences as follows:

   CR (0D) Carriage return causes data in the printer buffer to be printed. The print head is left in its current position.

   LF (0A) Line feed causes data, if any, in the printer buffer to be printed and advances the paper one line.

   BEL (07) Bell results in a tone signal.

   BS (08) Backspace results in a backspace operation.

Table B-5. Line Printer Character Set (Continued)

Notes: (Continued)

HT (09) Horizontal Tab causes spaces to be entered in the printer buffer up to the next horizontal tab position.

DC1 (11) Select selects the printer, enabling it to receive data.

DC2, N (12, N) Tab to line causes the paper to advance to the line specified by N. N must be greater than current line position.

DC3 (13) Deselect deselects the printer, preventing it from receiving data.

DEL (7F) Delete clears the printer buffer of all characters.

NUL (00) Null terminates the tab setting sequence (see below), otherwise it is ignored.

DC4, N (14, N) Tab to column causes the carriage to advance to the column specified by N. N must be greater than the current column position, otherwise it will be ignored.

ESC, 1, N1, N2, N3, . . ., NK, NUL (1B, 1, N1, N2, . . ., NK, 00) Set vertical tabs clears all existing tabs and sets new tabs at lines N1, N2, . . ., and NK.

ESC, 2, N (1B, 2, N) Set form length sets the form length used by the FF command to N.

ESC, 3, N1, N2, . . ., NK, NUL (1B, 3, N1, N2, . . ., NK, 00) Set horizontal tab clears all existing horizontal tabs and sets new tabs at locations N1, N2, . . ., NK.

ESC, 4 (1B, 4) Sets paper drive system to 6 lines per inch.

ESC, 5 (1B, 5) Sets paper drive system to 8 lines per inch.

ESC, 6 (1B, 6) Sets print size to 10 characters per inch.

ESC, 7 (1B, 7) Sets print size to 16.5 characters per inch.

ESC, 8, M (1B, 8, M) Stores vertical format data, either vertical tab locations or form length (see note 1), into vertical format channel M (non-volatile internal printer memory).

ESC, 9, M (1B, 9, M) Recalls vertical format data from vertical format channel M into working memory. This command has the same effect as issuing a set vertical tabs command or set form length.

ESC, :, N (1B,3A,N) Converts printer from 132 position printer to one of N positions where $2 \leqslant N \leqslant 127$.

ESC, ; (1B, 3B) Returns printing to normal 132 positions.

All of the preceding commands may not apply if the printer does not have the particular option needed to execute the command. If the printer does not have the required hardware to execute the command, the command will be ignored.

N, N1, N2, etc, used in the ESC commands represent 7 bit binary numbers. If the parity option is selected on the printer, correct parity must be supplied here also. M is an ASCII number, where $1 \leqslant M \leqslant 8$. Commas are used in the preceding commands for separators only and are not to be included in the commands.

Table B-5. Line Printer Character Set (Continued)

Notes: (Continued)

SO (0E) SO causes elongated character printing. The elongated character must be sent as the first character of the line and must precede each line printed in elongated characters. Also, with elongated print only 66 characters per line are allowed. If more than 66 characters are sent, the excess are printed on the next line as standard, nonelongated characters.

VT (0B) Vertical Tab causes data, if any, in the buffer to be printed and advances the paper to the next vertical tab location or top of form, whichever comes first. If no vertical tabs are set, a VT command causes the paper to advance to the top of the form.

FF (0C) Form Feed causes data, if any, in the printer buffer to be printed and advances the paper to top of the next form. At Power Up, the lines per form is 66.

For a write ASCII operation, the line printer DSR filters all control characters in the range $0-1F_{16}$, except the following:

BEL	07	Bell
BS	08	Backspace
HT	09	Horizontal tab (converted to a space)
LF	0A	Line feed
FF	0C	Form feed
CR	0D	Carriage return
S0	0E	Elongated character
S1	0F	(ignored by the line printer)

If the user wishes to use the full control character set, the Write Direct (0C) operation must be used. No characters will be filtered on a Write Direct (0C) operation.

Note 1: The two options, vertical forms control and form length, are mutually exclusive. The printer may have either option but not both.

Table B-6. ASR 33 Teletypewriter Character Set

	Printer	Keyboard	Punch (ASCII)	Punch (Direct)
END-OF-RECORD	Depletion of character count	CR	Depletion of character count	Depletion of character count
END-OF-FILE	N/A	DC3	DC3	N/A

	Reader (ASCII)	Reader (Direct)
END-OF-RECORD	CR	DC3
END-OF-FILE	DC3	N/A

Character	Hexadecimal	Character	Hexadecimal	Character	Hexadecimal
Space	20	A	41	C	63
!	21	B	42	D	64
”	22	C	43	E	65
#	23	D	44	F	66
$	24	E	45	G	67
%	25	F	46	H	68
&	26	G	47	I	69
’	27	H	48	J	6A
(	28	I	49	K	6B
)	29	J	4A	L	6C
*	2A	K	4B	M	6D
+	2B	L	4C	N	6E
,	2C	M	4D	O	6F
–	2D	N	4E	P	70
.	2E	O	4F	Q	71
/	2F	P	50	R	72
0	30	Q	51	S	73
1	31	R	52	T	74
2	32	S	53	U	75
3	33	T	54	V	76
4	34	U	55	W	77
5	35	V	56	X	78
6	36	W	57	Y	79
7	37	X	58	Z	7A
8	38	Y	59	[	7B
9	39	Z	5A	/	7C
:	3A	[	5B	]	7D
;	3B	/	5C	↑	7E
<	3C	]	5D		
=	3D	↑	5E		
>	3E	←	5F		
?	3F	@	60		
@	40	A	61		
		B	62		

Table B-6. ASR 33 Teletypewriter Character Set (Continued)

Notes

Printer:

1. BS character (08_{16}) results in a backspace operation.
2. Hl character (09_{16}) results in printing a space.
3. LF character ($0A_{16}$) results in a line feed operation.
4. CR character ($0D_{16}$) results in a carriage return operation.
5. FF character ($0C_{16}$) results in eight line feed operation.
6. End-of-record occurs when specified number of characters have been printed.
7. Maximum buffer size is 72 characters.

Keyboard:

1. BS character (08_{16}) is returned to printer as LF and BS. Deletes most recently entered character in buffer.
2. HT character (09_{16}) is returned to printer as space. Character is placed in buffer.
3. LF character ($0D_{16}$) is returned to printer as CR, and character is not placed in buffer; terminates the record.
4. DEL character ($7F_{16}$) is returned to printer as a line feed and carriage return. Character deletes current input record.
5. Maximum buffer size is 72 characters.

Paper Tape Punch:

1. The end of block character sequence for ASCII is CR ($0D_{16}$), LF ($0A_{16}$), DC3 (13_{16}), NULL (00_{16}), Null (00_{16}), Null (00_{16}), Null (00_{16}) for Direct is DC3 (13_{16}), Null (00_{16}), Null (00_{16}), Null (00_{16}), Null (00_{16}). The end-of-file character sequence for ASCII is DC3 (13_{16}), CR ($0D_{16}$), LF ($0A_{16}$), DC3 (13_{16}), Null (00_{16}), Null (00_{16}), Null (00_{16}), Null (00_{16}) for Direct is not applicable.
2. The characters BEL (07_{16}), BS (08_{16}), HT (09_{16}), and FF ($0C_{16}$) are punched on tape unchanged.
3. The character CR ($0D_{16}$) is translated to ETB (17_{16}) and punched.
4. The character DC3 (13_{16}) may be placed within a record, but may not be the first character of the record other than the end-of-file record.
5. End-of-record occurs when SCB specified number of characters have been punched.
6. Direct mode punches all characters. If a punch-off is in the buffer, a punch-on is output immediately following the character.

Paper Tape Reader:

1. All null characters are ignored at the first of the record and the first nonnull character is the first valid character.
2. The character BEL (07_{16}), BS (08_{16}), HT (09_{16}) and FF ($0C_{16}$) are stored in the user's buffer.
3. The character ETB (17_{16}) is stored in the user's buffer as a CR ($0D_{16}$).
4. The character CR ($0D_{16}$) indicates end-of-record, and is not stored in the buffer.
5. The character DC3 (13_{16}) as the first character of a record indicates an end-of-file record. The system returns end-of-file status after positioning the tape at the beginning of the next record. The DC3 is not stored in the buffer.
6. In direct mode all characters are stored in the buffer until the buffer is full, then the program searches for a DC3 character to terminate.

APPENDIX C

USER-SUPPLIED MODULES

APPENDIX C

USER-SUPPLIED MODULES

C.1 INTRODUCTION

Texas Instruments supplies support modules for a large number of user options in a TX990 operating system. However, there are some user needs that require modules that are not available, which the user must provide. These modules are:

- DSR (including an interrupt routine) for a unique peripheral device
- Extended Operation routines for user's extended operations
- Supervisor call routines for user supervisor calls

The user-supplied modules execute in the privileged mode in the Model 990/10 Computer, and all instructions are available. In the Model 990/4 Computer, all instructions are also available. For information on writing each type of routine consult the TX990 System Documentation manual, part number 944776-9701.

NOTE

When writing modules to be linked with the operating system (i.e., DSRs, XOPs, SVCs, user tasks), do NOT use the AORG (absolute origin) assembler directive. The TXBOOT program cannot load a system with any absolute origins in it.

APPENDIX D

GLOSSARY

APPENDIX D

GLOSSARY

Allocation Unit – A unit of space allocation on the diskette used during file management. An allocation unit is equal to six sectors. There are 333 allocation units on a diskette.

Boot Program – A program that loads the Operating System into memory and starts the Operating System executing.

COMMON – An area of memory which may be coded by use of the system console keyboard (e.g., a 733 ASR, a 911 VDT, et al) or by means of a task-specified-code and then made accessible for use by a task through the Get COMMON Data address supervisor call. The size of the system COMMON memory area is determined by a system parameter specified when the system is generated.

Default-substitute – A substitute pathname, or field of a pathname, provided by some utility programs when the program or keyboard-entry does not supply the data.

Device Name Table – A table accessed by the File Management supervisor call to obtain the address of the Physical Device Table (PDT) corresponding to a device name. Contains all device names defined in the system and addresses of the PDTs for the devices.

Device Service Routine – A routine of the TX990 Operating System that controls I/O operations with a device.

DNT – Device Name Table.

DSR – Device Service Routine.

Dynamic Task Area – The area of memory into which tasks may be loaded and executed. Tasks can only be loaded by using the Operator Communication Package (OCP) or the TXDS control program.

End-of-file – A record in a file (either logically or physically) that marks the end of the file. The character sequences that denote end-of-file for the file-oriented supported devices are shown in Appendix B.

End-of-record – A character of a record that marks the end of the record. The characters that denote end-of-record for supported devices are shown in Appendix B.

EOF – End-of-file.

EOR – End-of-record.

GENTX – The system generation task, which obtains system parameters interactively from the keyboard of the system console. GENTX builds source statement files from which modules TXDATA and TASKDF are assembled.

IDT – Program identifier of the source module.

Initial Program Load – The loading of a TX990 system from disc, cassette, or cards, placing the module in memory and starting execution of the system.

I/O Supervisor – The portion of TX990 that processes File Management supervisor calls, and passes control to the Device Service Routine (DSR) for the device.

IPL – Initial Program Load.

Keyboard Status Block (KSB) – A data structure in TXDATA used for character mode I/O with a VDT. TXDATA includes a KSB for each VDT.

KSB – Keyboard Status Block.

LDT – Logical Device Table.

Logical Device Table (LDT) – A table in TXDATA that contains a Logical Unit Number (LUNO) and the address of the Physical Device Table (PDT) that corresponds to the device assigned to the LUNO.

Logical Unit Number (LUNO) – A number by which an I/O operation specifies the device for the operation.

LUNO – Logical Unit Number.

OCP – Operator Communication Package.

Operator Communication Package (OCP) – A package of modules that contains the routines for the commands by which the operator or user communicates with TX990.

PC – Program Counter.

PDT – Physical Device Table.

Physical Device Table (PDT) – A table in TXDATA that contains device-related data required by the Device Service Routine (DSR) in an I/O supervisor call for the device.

Program Counter (PC) – A register in the computer hardware that contains the address of the next instruction to be executed.

Stand-alone Program – A program that executes without an operating system.

Status Register – A register in the computer hardware that contains condition bits and the interrupt mask.

Supervisor Call Block – A block of memory that defines a supervisor call, addressed by the XOP instruction. The code of the supervisor call is in byte 0 of the supervisor call block. The number of additional bytes (if any) and the content of the additional bytes are defined for each supervisor call.

SCB – Supervisor Call Block.

Supervisor Call Table – A table in TXROOT in which entry points to supervisor call routines are listed in a supervisor call code order.

Task Data Division – One of two logical divisions within a task. The data division contains one or more workspaces, data structures, supervisor call blocks, and data for the task. A data division may or may not be assembled separately from the procedure division of the task, and is not shared with any other task.

Task Management – Task Management maintains a state code for each task. The state codes are listed in Appendix G.

Task Scheduler – Initiates execution of a user task (see paragraph 11.2). When the currently executing task completes a time slice, the task scheduler passes control to the oldest task on the active list for the highest priority (0). If there is no task on the active list for priority 0, the oldest task on the active list for the next highest priority receives control.

Task Status Block (TSB) – A data structure in TXDATA used by the TX990 Operating System to control execution of the task.

Task Time Delay – The result of a task executing a Time Delay supervisor call. The Time Delay supervisor call suspends the calling task for a specified number of 50 ms periods.

Task Time Slice – A period of execution of a task having a maximum length defined when the system is generated. A task time slice begins when the task scheduler passes control to the task. A task time slice ends: (1) when the system suspends the task upon expiration of the maximum time period allowed for a task time slice; (2) when the task executes a supervisor call that suspends the task; (3) when the system suspends the task to await completion of an I/O operation. To avoid completely locking out low priority tasks, there is a maximum number of consecutive time slices (weighting factor) for each priority level. When the number of time slices has been used by a priority level, the oldest task on the active list for the next lower priority is allowed a time slice before the higher level again has control. The maximum number of time slices for each priority level are system parameters defined when the system is generated. The maximum period of a time slice may be extended by execution of a Do Not Suspend supervisor call. The time slice is less than the maximum time period when the task suspends itself, or is suspended awaiting completion of an I/O operation.

Task Weighting Factor – A count of task time slices for a priority level. When the number of task time slices specified as the weighting factor for priority level has been used by tasks at that priority level after a task at a lower level has had control, a task at a lower priority level receives control for a time slice.

Task Area, Dynamic – Memory area where tasks may be loaded and executed (see Dynamic Task Area and Task, Uses, Loading of).

Task, Active – A task which is in memory on an active queue, waiting for a time slice.

Task, Bid – To start execution of a task causing the TX990 Operating System to enter the task on the active list according to its priority level.

Task, Debugging of a – The process of removing errors from a task.

Task, Diagnostic (DTASK) – A system task that terminates a task when fatal errors occur in the task, and prints an error message.

Task, Executing a – Controlling the processor and the resources of the computer.

Task, Linked – Consists of separately assembled modules that have been combined by resolving external references and definitions in the modules to form a single executable module.

Task, Loaded – A task copied from an external storage medium into the memory of the computer in preparation for execution.

Tasks, Multiple – Two or more tasks concurrently active in an operating system.

Task, Procedure Division – One of two logical divisions within a task. The procedure division contains the executable code for the task. A procedure division may or may not be assembled separately from the data division of the task and may be shared with other tasks.

Task, Suspended – A task temporarily removed from the active list and from execution as a result of a supervisor call or during an I/O operation.

Task, Terminated – A task removed from execution and from the active list either at normal completion or at an abnormal termination initiated by the operator or by the diagnostic task when a fatal error is detected.

Task, Time Delay – A task which has halted execution for a specified length of time, at the end of which the task is reactivated.

Task, User, Loading of – The task loaded into the dynamic task area using the OCP LPROG command.

Task, Waiting – A task waiting for completion of an I/O operation or for a system function or resource.

Volume Name – A one to four character name, other than a drive name, by which a diskette may be accessed.

Workspace – A 16-word area of memory addressed as workspace registers 0 through 15. The active workspace is defined by the contents of the workspace pointer register.

Workspace Pointer (WP) – A register that contains the address of workspace register 0.

Workspace Register – A memory word accessible to an instruction of the computer as a general purpose register. It may be used as an accumulator, a data register, an index register, or an address register.

WP – Workspace pointer register.

APPENDIX E

TX990 - DX10 COMPATIBILITY

APPENDIX E

TX990 - DX10 COMPATIBILITY

E.1 INTRODUCTION

The upward compatibility of the TX990 operating system to DX10 operating system (release series 3.0 and higher) allows tasks that execute under TX990 to be executed under DX10 when the proper considerations are met. This appendix describes those considerations. This appendix also compares the supervisor calls of the two systems. The compatibility information cannot be specified because of the possible existence of several releases of each system. For information about compatibility with a specific release of DX10, the user must study the information about DX10 and note the difference.

E.2 TASK STRUCTURE COMPATIBILITY

The task structure required for the TX990 operation system is compatible with that of DX10. Tasks that execute in the dynamic task area of TX990 and other tasks that consist of a data division and procedure division in the same module may be installed as either memory-resident or disc-resident tasks under DX10 without modification. Tasks that share a procedure division use the mapping capabilities of the Model990/10 Computer when executed under the DX10 operating system. The addresses of the procedure division are mapped into the low order addresses of the addressable memory, and the addresses of the data division are mapped into the addresses above those of the procedure, beginning on a 32-byte boundary. Only the address in the index register used to address data is affected by this difference. The task may be relinked to provide this address, or the address can be patched.

E.3 SUPERVISOR CALLS

The following supervisor calls are available to the TX990 operating system user but are not supported by DX10.

- VDT Utility
- Make Task Privileged
- VDT Get Character
- Get Own ID
- VDT Conditional Get Character
- Get System Table

The following supervisor calls that perform no operational activity in the TX990 operating system perform operations in the DX10 operating system as described in the DX10 operating system manuals.

- Return Memory
- Return Common

The End of Program supervisor call performs different operations in the TX990 operating system; after terminating the task, it activates the rebid task, which is normally the TX990 Control Program.

The TX990 Operating System task that shares a procedure and contains both a Get Memory and a Get Common Data Address supervisor call has an additional restriction under DX10. Memory management in DX10 utilizes the mapping capabilities of the computer. The addressable memory area of a task consists of three contiguous address segments. The procedure division is mapped into the lowest of these address segments, beginning at address zero. The data division is mapped into the lowest of these address segments, beginning at address zero. The data division is mapped into the next higher segment, leaving one segment for either common memory or a memory block obtained with a Get Memory call. A Get Memory call must be followed by a Release Memory call before a Get Common Data Address call can be executed. Similarly, a Get Common Data Address call must be followed by a Return Common call before a Get Memory call can be executed. The address returned by a Get Memory supervisor call or a Get Common Data address is aligned on a 32-byte boundary in DX10.

E.4 FILE MANAGEMENT

The TX990 Operating System supports two file types, sequential and relative record. Sequential files are automatically blocked and blank compressed in the TX990 Operating System while in the DX10 Operating System the user optionally controls blocking and blank compression. Only one user may access a sequential file at a time; the user's position in the file is maintained across close and assign operations while in the DX10 Operating System multiple reads and a single write operation may be operating simultaneously, and assign operations cause a rewind operation to occur. The file pathname, under a TX990 Operating System assign operation, is compatible with the DX10 Operating System if the following structure is maintained; device.file name (where device is a maximum of 4 characters and file name is a maximum of 6 characters, e.g., DSC.FILMN). The 'exclusive all' access privilege code on an open service call in the TX990 Operation System is 00 while in the DX10 Operating System it is an 01.

APPENDIX F

COMPRESSED OBJECT CODE FORMAT

APPENDIX F

COMPRESSED OBJECT CODE FORMAT

The standard object code format under the TX990 Operating System is comprised basically of an ASCII tag character followed by one or two ASCII fields. The first field is numeric in value and the optional second field contains a symbol. (For additional familiarity with standard object code format, refer to Section 9.5 of the Model 990 Computer Assembly Language Programmer's Guide, part number 943441-9701.) The first ASCII field in standard object code format is four characters (i.e., four bytes) in length which, when converted to compressed object code format, is changed to binary, two bytes in length. The second field in standard object code format is left unchanged when converting to compressed object code format. Records are terminated with the standard end-of-record tag character, only. The beginning-of-module-tag-character is an ASCII zero in standard object code format and a binary one in compressed object code format. This is used to distinguish between compressed and uncompressed modules. The end-of-module colon record, identified by the colon at the beginning of the last line of the module, is unchanged. The diskette is the only device capable of supporting compressed object code format.

ASCII Standard Object Code Format (e.g., from punched cards)

```
00008TASK          A0000B000AB020000000B00007F7EEF
:       TASK              021/77      12:32:54
```

Hexadecimal Representation of Standard Format

```
3030  3030  3854  4153  4B20  2020  2041  3030
3030  4230  3030  4142  3032  3030  4330  3030
3042  4330  3030  3746  3745  4546  2020  2020
```

ASCII Representation of Standard Format

```
00  00  8T  AS  K       A   00
00  B0  00  AB  02  00  C0  00
0B  C0  00  7F  7E  EF
```

Hexadecimal Representation of Compressed Format

```
0100  0854  4153  4B20  2020  2041  0000  4200
0A42  0200  4300  0042  C000  4600  0000  0000
```

ASCII Representation of Compressed Format

```
..  .T  AS  K       A   ..  B.
.B  ..  C.  .B  ..  F.  ..  ..
```

Colon Record for Both Formats Hexadecimal Representation

```
3A20  2020  2020  2054  4153  4B20  2020  2020
2030  3231  2F37  3720  2020  2031  323A  3332
3A35  3420  2020
```

ASCII Representation

```
:       T  AS  K
 0  21  /77      1  2:  32
:5  4   .
```

APPENDIX G

TASK STATE CODES

APPENDIX G

TASK STATE CODES

The user-task supervisor calls which return one of the task state codes listed in table G-1 to byte 1 of the supervisor call block are:

- Bid Task Supervisor Call
- Activate Suspended Task Supervisor Call
- Activate Time Delay Task Supervisor Call

The user may code his program to read out the task state code to an output device or, using the OCP STate (ST) command, the user can cause a terminal to print out the task state codes.

Table G-1. List of Task State Codes

Code (Hexadecimal)	Significance
00	Active task, priority level 0
01	Active task, priority level 1
02	Active task, priority level 2
03	Active task, priority level 3
04	Terminated task
05	Task in time delay
06	Suspended task
07	Currently executing task
08	Task awaiting VDT character input
09	Task awaiting completion of I/O
0A	Task queued for I/O
0B	Task queued for file utility routine
0C	Task on the diagnostic queue
0D	Task waiting for file management completion
10	Task queued for file management

APPENDIX H

PRINTOUT OF FATAL TASK ERROR CODES OR DISPLAY OF ILLEGAL INTERRUPT CODE

APPENDIX H

PRINTOUT OF FATAL TASK ERROR CODES OR DISPLAY OF ILLEGAL INTERRUPT CODE

The capability of printing out fatal task error codes or displaying an illegal interrupt code is available to the user when the Diagnostic Task (DTASK) module is supplied with the TX990 Operating System. DTASK begins to execute when the TX990 Operating System error logic detects a fatal error during task execution; the result of the DTASK module being executed is the printout of the fatal task error code or the display of the illegal interrupt code. The fatal task error code is printed out on a LUNO, as described below in this appendix, when the task being executed does not specify an end action to be executed; the illegal interrupt is displayed on the Programmer Panel. When DTASK prints the error message identifying the error, it causes the task to be terminated. The fatal errors detected by TX990 are listed in table H-1.

When TX990 detects that execution of a task has resulted in a fatal error and the task does not perform end action, the error logic removes the task in error from the active queue, places it on the diagnostic queue, sets the task's state to on diagnostic queue, and executes DTASK. DTASK prints an error message on LUNO 0 as shown in the following example:

```
*TASK=0F,ERROR=03,ADDRESS-214E
WP=2154,PC-28A0,ST=300F
```

The example shows that the task having identifier F_{16} attempted to access memory address beyond the existing range of memory (error code 3). The task address is $214E_{16}$; the current workspace is at address 2154_{16}; and the program counter contains $28A0_{16}$. Since the program counter is advanced to the next address before an instruction is executed, the instruction that attempted the memory access would be at address $289A_{16}$, $289C_{16}$, or $289E_{16}$, according to the size of the instruction. The status register contains $300F_{16}$.

After printing the error message, DTASK sets a flag in the Task Status Block (TSB) and places the task on the active queue. The flag set by DTASK causes the scheduler to terminate the task.

In the event that DTASK itself commits a fatal error, it prints the following message on LUNO 0:

```
HELP!
```

This represents a system error that requires the system to be loaded again.

Table H-1. Fatal Task Error Codes and Illegal Interrupt Code

Error Code	Error Description
1	Memory parity error. Can only occur if the computer is equipped with memory parity or error correction hardware.
2	Illegal operation code. Can only occur in a Model 990/10 Computer.
3	TILINE* timeout. Nonexistent memory was addressed. Can only occur in a Model 990/10 Computer.
4	Illegal XOP. An XOP instruction having an undefined extended operation number or an illegal supervisor call was executed.
5	Map error. An access outside of the address space defined by the current map was attempted. Can occur only in a Model 990/10 Computer with map option in which the map is enabled. Since TX990 does not enable the map, this error could only occur if the user task enabled the map.
6	Privileged operation. A privileged instruction was attempted in the nonprivileged mode. Can only occur in a Model 990/10 Computer in the nonprivileged mode.
7	Illegal interrupt. An external interrupt, for which there is no interrupt routine, has occurred. The user may display the status register to determine the interrupt level and must clear the interrupt before attempting to resume operation.

*TILINE is a registered trademark of Texas Instruments Incorporated.

APPENDIX I

I/O ERROR CODES

APPENDIX I

I/O ERROR CODES

Code (Hexadecimal)	Description
	DSR ERRORS
00	NO ERROR
01	ILLEGAL LUNO
02	ILLEGAL OPERATION CODE
03	LUNO IS NOT YET OPENED
04	RECORD LOST DUE TO POWER FAILURE
05	ILLEGAL MEMORY ADDRESS
06	TIME OUT, OR ABORT
07	READ CHECK ERROR
11	DEVICE ERROR
12	NO ADDRESS MARK FOUND
15	DATA CHECK ERROR
19	DISKETTE NOT READY
1A	WRITE PROTECT
1B	EQUIPMENT CHECK ERROR
1C	INVALID TRACK OR SECTOR
1D	SEEK ERROR OR ID NOT FOUND
1E	DELETED SECTOR DETECTED
	FILE MANAGEMENT ERRORS
20	LUNO IS IN USE
21	BAD DISC NAME
22	PATHNAME HAS A SYNTAX ERROR
23	ILLEGAL FUR OPCODE
24	BAD PARAMETER IN PRB
25	DISKETTE IS FULL
26	DUPLICATE FILE NAME
27	FILE NAME IS UNDEFINED
28	ILLEGAL LUNO
29	SYSTEM BUFFER AREA FULL
2A	SYSTEM CAN'T GET MEMORY
2B	FILE MANAGEMENT ERROR
2C	CAN'T RELEASE SYSTEM LUNO
2D	FILE IS PROTECTED
2E	ABNORMAL FUR TERMINATION
2F	FILE UTILITY DOESN'T EXIST IN SYSTEM
30	NON-EXISTENT RECORD
3B	INVALID ACCESS PRIVILEGE
3E	FILE CONTROL BLOCK ERROR
3F	FILE DIRECTORY FULL

I/O ERROR CODES (Continued)

Code (Hexadecimal)	Description
	TASK LOADER ERROR
60	I/O ERROR, LOAD NOT COMPLETE
61	OBJECT MODULE CONTAINS NONRELOCATABLE OBJECT CODE
62	CHECKSUM ERROR LOAD ABORTED
63	LOADER RAN OUT OF MEMORY
64	TASK 10 IS BUSY
65	IMAGE FILE ERROR
	VDT ERRORS
80	DEVICE NOT AVAILABLE VDT STATION NOT FOUND

Note:

Error Code >FF is a general error code.

APPENDIX J

SYSTEM TASKS

APPENDIX J

SYSTEM TASKS

J.1 INTRODUCTION

There are eight system tasks in a complete TX990 Operating System: (1) one Operator Communication Package (OCP); (2) one Initial Start Task (STASK); (3) one Diagnostic Task (DTASK); (4) four File Management Tasks (TXFMP1, TXFMP2, TXFMP3, TXFMP4); (5) one File Utility Routine (FUR); (6) one rebid task. The operation of OCP and the supported OCP commands are described in Section V. Descriptions of the other system tasks and their operation are included in this appendix.

J.2 INITIAL START TASK (STASK)

The initial start task (STASK) begins execution at the completion of each IPL to identify the revision level of the operating system and to indicate that the IPL has executed successfully. The following is an example of the message printed by STASK:

```
TX990 SYSTEM                          Release 2.2
MEMORY SIZE WORDS: 12288 AVAILABLE: 5940
```

The memory size printed is the decimal number of words of memory in the system. The size, in words, of the dynamic task area, is printed as the available memory.

When STASK is linked following TXEND, it is located in the dynamic task area, where it will be overlaid by the first user task installed in the dynamic task area. It is assigned task identifier 10_{16}. When it is desired to execute STASK following a manual restart, STASK must be linked ahead of TXEND.

J.3 DIAGNOSTIC TASK (DTASK – TASK ID D_{16}).

The diagnostic task (DTASK) is executed when TX990 error logic detects a fatal error in a task. DTASK prints an error message identifying the error and causes the task to be terminated in error. The fatal errors detected by TX990 are listed in Appendix F.

When TX990 detects that execution of a task has resulted in a fatal error and the task does not perform end action, the error logic removes the task in error from the active queue, places it on the diagnostic queue, sets the task's state to on diagnostic queue, and executes DTASK. DTASK prints an error message on LUNO 0 as shown in the following example:

```
*TASK=0F, ERROR=03, ADDRESS=214E
WP=2154, PC=28A0, ST=300F
```

The example shows that the task having identifier F_{16} attempted to access a memory address beyond the existing range of memory (error code 3). The task address is $214E_{16}$, the current workspace is at address 2154_{16}, and the program counter contains $28A0_{16}$. Since the program counter is advanced to the next address before an instruction is executed, the instruction that attempted the memory access would be at address $289A_{16}$, $289C_{16}$, or $289F_{16}$, according to the size of the instruction. The status register contains $300F_{16}$.

After printing the error message, DTASK sets a flag in the Task Status Block (TSB) and places the task on the active queue. The flag set by DTASK causes the scheduler to terminate the task.

In the event that DTASK itself commits a fatal error, it prints the following message on LUNO 0:

HELP!

This represents a system error that requires the system to be loaded again.

J.4 FILE MANAGEMENT TASKS (TXFMP1, TXFMP2, TXFMP3, TXFMP4 – TASK ID, $F0_{16}$, $F1_{16}$, $F2_{16}$, $F3_{16}$)

File management must be included in the TX990/TXDS system to enable the user task to communicate with files on the diskette. File management consists of one procedure and a data section for each diskette drive included in the system. The four data sections (TXFMP1, TXFMP2, TXFMP3, TXFMP4) when linked with the procedure (TXFMP) define the File Management task and must be included in the task definition at system generation time. (See Section 7 entitled "System Generation".) If one diskette drive is in the system, TXFMP1 has to be included: if two diskette drives are included in the system, both TXFMP1 and one TXFMP2 must be included: etc. The required GENTX parameters for these file management tasks are described in the System Task Definition Table.

J.5 FILE UTILITY TASK (FUR – TASK ID B_{16})

The File Utility Routine (FUR) assigns and releases LUNOs to files and performs file maintenance functions.

If the file management task is included in the system, the File Utility task must also be included during system generation. The required GENTX parameters for FUR are described in table 7-1 entitled "System Task Definition".

J.6 REBID TASK

The system rebid task is the task that will be activated whenever an end program supervisor call is made. For standard TX990/TXDS systems, the TXDS control program is defined as the rebid task, and will be activated when a task terminates with an end program supervisor call, task ID 16_{16}. The user may wish to designate another task as the rebid task. This may be accomplished in the following manner:

1. Place the following statements in the new Rebid Task.

```
        DEF   REBID
REBID   EQU   ID*>100
```

   where ID = Task ID of the new Rebid task.

2. If the TXDS control program is included in the system, link this module before module CNTROL.

J.7 VOLUME NAME SUPPORT (VOLUME – TASK ID C_{16})

The volume name support task associates a volume name with a diskette drive, allowing files on the diskette to be assessed through the volume name as well as the device name.

APPENDIX K

TX990 SYSTEM GENERATION USING DX10 RELEASE 3.0

APPENDIX K

TX990 SYSTEM GENERATION USING DX10 RELEASE 3.0

GENTX must be installed and executed using the IT and XTS commands on a DX10 system.

Perform the following steps on a DX10 release 3.0 system:

1. Place the GENTX object module on a DX10 sequential file or device. If the module is on cassette, place it in a cassette drive. If it is on diskette, execute the DX10 conversion utility TXDX and convert the object file to a DX10 sequential file.

2. Install the task, using the IT command:

   II PF = .S$PROGA, TN = GENTX, OBJ = <acnm>, TI = <id>

   where <acnm> is the DX10 access name for the file or cassette unit which contains the GNTXDX object module and <id> is the task's ID

3. Execute the task, using the XTS command:

   XTS TN = GENTX, PF = .S$PROGA

4. Upon initiation of the generation procedure, the following message is displayed by the GENTX task:

   TX990 SYSTEM GENERATION – 945673*C

5. After the message is displayed, the GENTX program enters the definition phase described in paragraphs 7.4.2 through 7.4.10. Refer to this section for a description of the user options.

6. Upon completion of the construction phase the source programs generated for TXDATA and TASKDF must be assembled. These source programs may be assembled using SDSMAC. The object code files created will be used as input to the linking process in step 10.

7. At this point, the TX990 object modules must be selected from the DSRLIB, OCPLIB, FMPLIB and TXLIB cassettes. This selection process is accomplished by use of the OBJMGR (Object Manager Task). A file is created as output from OBJMGR and will be used as input to the linking process in step 18.

8. The Link Editor is used to link the object modules of TXDATA, TASKDF, and TX990 parts created in the above steps. This may be accomplished using the following link control file:

```
NOSYMT
PHASE      0,TX990
INCLUDE    name 1
INCLUDE    name 2
INCLUDE    name 3
END
```

where

name 1 is the pathname for TXDATA object file

name 2 is the pathname for TASKDF object file

name 3 is the pathname for the selected TX990 parts file.

The output of SDSLNK may now be copied to cassette or diskette and used on a TX990 system.

APPENDIX L

SUPPORT FOR THE 5MT/6MT I/O INTERFACE SPECIAL DEVICE

APPENDIX L

SUPPORT FOR THE 5MT/6MT I/O INTERFACE SPECIAL DEVICE

L.1 SYSTEM GENERATION PARAMETERS

To include a 5MT/6MT Serial I/O Interface Module in a hardware configuration, a new operating system must be generated, as described in the section on system generation. During execution of GENTX, the 5MT/6MT module must be defined as a special device. The responses to GENTX prompts are shown in table L-1. Items in all capitals must be entered as shown.

Table L-1. GENTX Parameters for the 5MT/6MT Modules

Prompt	Response
CHASSIS	Number of the chassis in which the interface card is connected (0-7)
DEV NAME	1 to 4 character name (e.g. 1056); first character must be alphabetic
DEV TYPE	SD
CRU BASE ADDR	As shown in the chart on top of the chassis
ACCESS MODE	FILE
INT LEVEL or INT POSITION	As shown in the chart on top of the chassis As connected to the expansion chassis
TIME-OUT COUNT	Desired number, as described in section 9
CRU INT LINE	31
ENTRY LABEL OF DSR	DSR5MT
ENTRY LABEL OF ROUTINE	INTRUP
INT BRANCH LABEL	UNSOL
EXTENSION DATA	carriage return

L.2 INCLUDING THE DEVICE SERVICE ROUTINE

When linking the newly generated system, the module DSR5MT in file :DSRLIB/OBJ on the TX990 parts diskette must be included. See the paragraph entitled LINKING THE NEW SYSTEM in section 9, and the section on the Object Manager, OBJMGR, for details on how to include this DSR.

L.3 FUNCTIONAL DESCRIPTION OF THE 5MT/6MT MODULE

There are different modes of operation provided by the 5MT/6MT DSR: read status, read-random, write-random, read-sequential, and write-sequential mode. Any of the read/write modes may be executed in conjunction with a transmit/receive operation using the same supervisor call block (SCB), which is described in paragraph L.4.

All operations possible with the 5MT/6MT interface module can be performed using the SCB and its data base. No other control commands are necessary.

These various modes of operation allow the user to output or retrieve information in single bit format (up to 16 bits at a time) or single or multiple word format (up to 16 words at a time), and are described in the following paragraphs.

L.3.1 READ STATUS MODE. The read status mode allows the user to perform a self-test type of operation on the interface module.

When a read status call is made the DSR causes the interface to dump the contents of its output RAM into its input RAM and then read the input RAM into a 16 word program buffer whose address is specified in the SCB.

Each time a read status call is made the DSR will place several status flags in the 17th word of the program buffer. The flag bits have the following significance when set:

Bit	Meaning
0-8	Not used
9	Self-test enabled
10-11	If 00 – transmit/receive mode 01 – sequential read/write mode 11 – random read/write mode
12	I/F module busy
13	Power on (0 = power off)
14	I/F interrupt unmasked
15	I/F interrupt present

The usefulness of this call is that the user could output a set of known test data to the output RAM and then compare it with the information retrieved from the input buffer to see if the two match. If the two sets of data do not match then this indicates the possibility of a hardware problem.

L.3.2 READ-RANDOM MODE. The read-random mode retrieves information from the input RAM one bit at a time or up to as many as 16 bits at a time, and is performed after any transmit/receive cycle.

The SCB provides an address in the calling program for: storage of the input information; the starting address of the input RAM where the information will be taken from; and a bit count indicating the number of bits to be input.

With this operation, the status of a single bit can be tested.

The information input during a read-random operation is always placed by the DSR in a one-word program buffer with the starting RAM bit beginning at the right-most bit position of that word. Any bit positions to the left not filled with information will be filled with zeroes.

L.3.3 WRITE-RANDOM MODE. The write-random mode outputs information from a buffer to the output RAM on the 5MT/6MT serial interface, one bit at a time, or up to 16 bits at a time.

The information to be output is stored in a calling program buffer whose address is given in the SCB of the calling program. The SCB also provides the starting address of a location within the calling program buffer. The DSR starts at this location and outputs the number of bits specified in the bit count of the SCB, to the 5MT/6MT output RAM.

With this operation, a single bit can be output at a time.

The write-random operation is performed before any transmit/receive operation.

L.3.4 READ-SEQUENTIAL MODE. The read-sequential mode retrieves information from the input RAM in the form of 16 bit words. The read-sequential mode can input as many as 16 words (256 bits) at a time.

The SCB provides an address pointing to a buffer in the calling program where the input information will be stored; a RAM base address, pointing to one of the 16 words in RAM where the information will be taken from; and a count indicating the number of words to be retrieved.

The read-sequential operation is performed after the transmit/receive operation.

L.3.5 WRITE-SEQUENTIAL MODE. The write-sequential mode outputs information from a buffer to the output RAM of the 5MT/6MT serial interface, in 16 bit word format, and up to as many as 16 words at a time. The information to be output is stored in a calling program buffer whose address is given in the SCB of the calling program. The SCB also provides the address which points to a starting location in the calling program buffer where the information to be output is taken from. The DSR starts at this location and outputs the number of bytes specified in the character count of the SCB to the output RAM.

L.3.6 TRANSMIT/RECEIVE OPERATION. The transmit/receive operation allows the user to output all of the information contained in the output RAM to the 5MT/6MT modules while reading the present status of the modules into the input RAM of the serial interface. Both operations are performed at the same time.

The transmit/receive (T/R) operation is initiated by clearing a bit in byte 5 of the SCB.

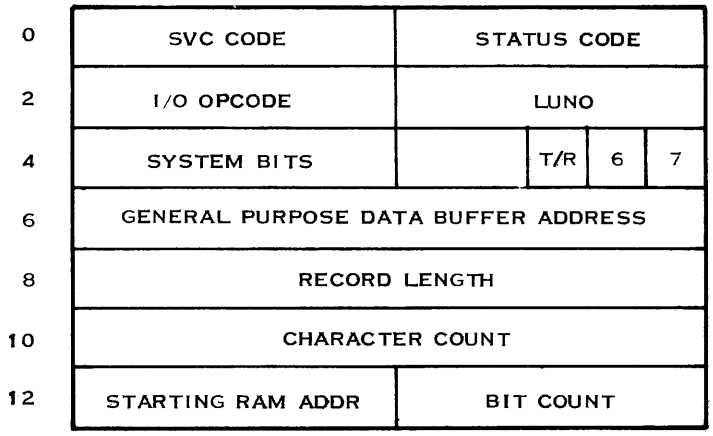
When the T/R operation is completed an interrupt is generated. The calling task is suspended until the interrupt is received unless the initiate bit in byte 4 of the SCB is set. When the initiate bit is set, control is returned to the calling task after I/O is initiated. When an interrupt is received the information from the 5MT/6MT modules is then present in the input RAM and can be stored in the input sotrage buffer.

The transmit/receive operation precedes the read operations and follows the write operations when the T/R bit is set.

L.4 SUPERVISOR CALL BLOCK FORMAT

The SCB consists of a 7 word block as shown in figure L-1. The first six words in the block are standard for all types of I/O with the exception of byte 5 which contains user set bits peculiar to this type of I/O. The remaining word belongs exclusively to this type of I/O.

Byte Ø contains the supervisor call code (ØØ) for I/O operation.



(A)137507

Figure L-1. 5MT/6MT I/O Call Block

Byte 1 is the reserved location where the system places a status code after completion of the operation.

There are 4 status codes that may be returned by the system.

Normal Completion (ØØ)
Illegal Opertion (Ø2)
Record Loss Due To Power Failure (Ø4)
Operation Timed Out Or Terminated Abnormally (Ø6)

Byte 2 of the SCB contains the user placed I/O opcode indicating which I/O operation will be performed.

The I/O operations are:

- OPEN (ØØ). The 5MT/6MT serial interface module shall be considered a file oriented device, so that an 'OPEN' operation should be performed before proceeding with any other operations. The 'OPEN' call returns a status code of ØØ in byte 1 of the SCB.

- CLOSE (Ø1). The 'CLOSE' call ends I/O operation to the device and places a code of ØØ in the status byte of the SCB.

- READ STATUS (Ø5)

- READ-SEQUENTIAL (20_{16})

- READ-RANDOM (21_{16})
- WRITE-SEQUENTIAL (22_{16})
- WRITE-RANDOM (23_{16})

Byte 3 contains the logical unit number (LUNO).

Byte 4 contains the system flags, described in detail in section 7.

Byte 5 contains the transmit/receive (T/R) bit in bit location 5. This bit initiates the transmit/receive cycle when reset. The cycle being automatically after a write operation and before any read operation.

Bytes 6-7 General Purpose Data Buffer Address. This location can serve as two different types of addresses depending on the type of I/O call being made.

For a read call the value stored here is the input data buffer address. This address points to the location in the calling program where the data to be input from the modules is stored.

For a write call the value stored here is the output data buffer address. This address points to the location in the calling program where the data to be output to the modules is stored.

Bytes 8-9 Record Length. This location indicates the maximum number of bytes to be transferred during a read sequential operation.

Bytes 10-11 Character Count. The system returns the actual number of bytes transferred during a read sequential operation in this location. For a write sequential operation this location contains the number of bytes to be output to the module.

Byte 12 Starting Location Address. The value placed in this location applies to both read and write operations.

For read operations this points to the starting location in the input RAM where the input operation is begun. For bit input, the value in this byte ranges from 0 to 255. For words input, the value ranges from 0 to 15. During the read operation the DSR looks at the starting location address value, goes to that corresponding address on the input RAM and inputs the number of words or bits specified by the words/bits count into the calling program buffer. The words or bits are placed in the program buffer in locations which correspond to the RAM locations.

Byte 13 Bit Count. This location is used by random-type I/O only. The value placed here must be between 0 and 15 and indicates how many bits are to be input or output. A value of 0 indicates that 16 bits are to be input or output.

APPENDIX M

SUPPORT FOR THE 32-IN/TRANSITION DETECTION MODULE SPECIAL DEVICE

APPENDIX M

SUPPORT FOR THE 32-IN/TRANSITION DETECTION MODULE SPECIAL DEVICE

M.1 SYSTEM GENERATION PARAMETERS

The 32-In/Transition Detection module may be supported as a special device by a customized TX990 operating system. To generate a system which includes the module, initiate the system generation program GENTX, as described in section 9. During the device definition phase, define the module as a special device by responding to the GENTX prompts as shown in table M-1.

Table M-1. GENTX Parameters for the 32-In/Module

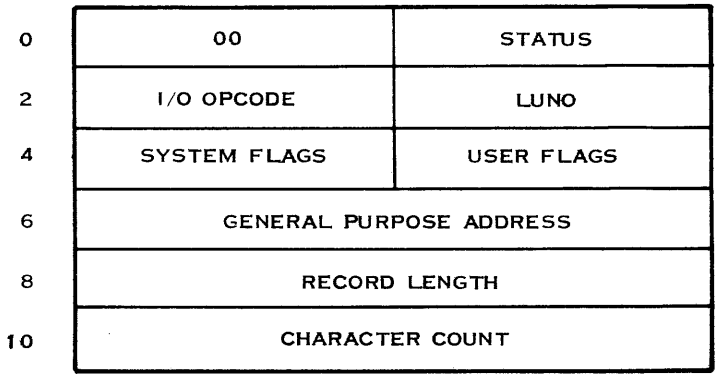
Prompt	Response
CHASSIS	Number of the chassis to which the module is connected (0-7)
DEV NAME	1 to 4 character name (e.g., SD32)
DEV TYPE	SD
CRU BASE ADDR	As shown in the chart on top of the chassis
ACCESS MODE	FILE
INT LEVEL	As shown in chart on top of main chassis
or	
INT POSITION	As shown in chart on expansion chassis
TIME-OUT COUNT	Desired number, as described in section 9
CRU INT LINE	31
ENTRY LABEL OF DSR	DIGDSR
ENTRY LABEL OF ROUTINE	INT321
INT BRANCH LABEL	NONSOL
EXTENSION DATA	carriage return

M.2 INCLUDING THE DEVICE SERVICE ROUTINE

When linking the object modules of the new operating system together, the module DIGDSR in file :DSRLIB/OBJ on the TX990 parts diskette must be included.

M.3 SUPERVISOR CALL BLOCK FORMAT

Figure M-1 shows the format of the I/O call block used for all 32-In/Transition module operations.



(A)137508

Figure M-1. 32-In/T Call Block Format

The first three words of the block are standard for all I/O supervisor calls:

Byte 0 contains the supervisor call code (ØØ) for I/O operation.

Byte 1 is the reserved location where the system places a status code after completion of the operation.

There are 4 status codes that may be returned by the system:

ØØ – Normal Completion
Ø2 – Illegal Operation Code
Ø4 – Recorded Loss Due To Power Failure
Ø6 – Operation Timed Out Or Terminated Abnormally

Byte 2 of the SCB contains the user placed I/O operation code:

- OPEN (ØØ) – The 32-In/Transition module is considered a file oriented device; therefore, an 'OPEN' operation should be performed before proceeding with any other operations. The 'OPEN' returns a status code of 0 in byte 1 of the SVC block.

- CLOSE (Ø1) – The 'CLOSE' call ends I/O operation to the device and places a code of Ø in the status byte of the SVC block.

- READ 32 BITS ($2Ø_{16}$) – The 'READ 32 BITS' call reads 32 bits of information from the module and places it at the location given by the address contained in bytes 6 and 7 of the SCB.

- WRITE MASK (22_{16}) – The 'WRITE MASK' call outputs 32 bits of information located at the address given in bytes 6 and 7 of the SVC to the module. This data acts as a bit level interrupt mask. When a '1' is output to a particular line on the module this enables a board level interrupt. The computer however, does not recognize these board level interrupts until a 'READ INTERRUPTING BIT' call is executed. This call allows the computer to recognize any unmasked board interrupts according to the data in the 'WRITE MASK' call.

- READ INTERRUPTING BIT (21_{16}) – The 'READ INTERRUPTING BIT' call enabled the computer to recognize any board level interrupts set up previously by the 'WRITE MASK' I/O call.

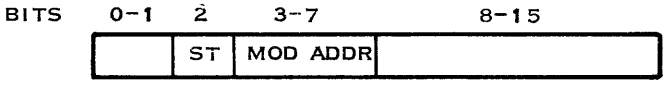
Byte 3 contains the 'LOGICAL UNIT NUMBER' (LUNO).

Bytes 4 and 5 contain the system flags and user flags described in Section VII.

Bytes 6 and 7 form a general purpose address which can be used for three separate operations. It provides for the (1) address of a two word mask, (2) the address of the interrupt data, and (3) the address of a two word block of data. These three addresses are described in the following paragraphs.

When used with the 'WRITE MASK' I/O call, bytes 6 and 7 contain the address of a two word mask for enabling or disabling module interrupts. Any bits set to a '1' by the two word mask will enable a corresponding interrupt on the module. A bit reset to a 'Ø' will prevent an interrupt from occurring.

When used with a 'READ INTERRUPTING BIT' I/O call the module address of the interrupt data is returned by the system at the address given in bytes 6 and 7. The returned module address appears in the left-most byte of the word whose address is given in bytes 6 and 7, along with the status (1 or 0) of the particular line on the module that caused the interrupt. The configuration of the returned module data looks like this:



(A)137509

When used with a 'READ 32 BITS' I/O call, bytes 6 and 7 contain the address pointing to the location where the 32 bits of information are stored after the operation.

Bytes 8 and 9 contain the maximum number of bytes to be transferred during any operation, therefore, a value of 4 must be placed here by the user during all operations.

Bytes 10 and 11 contain the actual number of bytes transferred by the system. A value of 4 will be placed here by the DSR during any operation.

ALPHABETICAL INDEX

ALPHABETICAL INDEX

INTRODUCTION

The following index lists key words and concepts from the subject material of the manual together with the area(s) in the manual that supply major coverage of the listed concept. The numbers along the right side of the listing reference the following manual areas:

- Sections - References to Sections of the manual appear as "Section x" with the symbol x representing any numeric quantity.
- Appendixes - References to Appendixes of the manual appear as "Appendix y" with the symbol y representing any capital letter.
- Paragraphs - References to paragraphs of the manual appear as a series of alphanumeric or numeric characters punctuated with decimal points. Only the first character of the string may be a letter; all subsequent characters are numbers. The first character refers to the section or appendix of the manual in which the paragraph is found.
- Tables - References to tables in the manual are represented by the capital letter T followed immediately by another alphanumeric character (representing the section or appendix of the manual containing the table). The second character is followed by a dash (-) and a number:

  Tx-yy

- Figures - References to figures in the manual are represented by the capital letter F followed immediately by another alphanumeric character (representing the section or appendix of the manual containing the figure). The second character is followed by a dash (-) and a number:

  Fx-yy

- Other entries in the Index - References to other entries in the index are preceded by the word "See" followed by the referenced entry.

USER'S RESPONSE SHEET

Manual Title: Model 990 Computer TX990 Operating System Programmer's Guide (Release 2) (946259-9701)

Manual Date: 15 December 1977 **Date of This Letter:** _____

User's Name: _____ **Telephone:** _____

Company: _____ **Office/Department:** _____

Street Address: _____

City/State/Zip Code: _____

Please list any discrepancy found in this manual by page, paragraph, figure, or table number in the following space. If there are any other suggestions that you wish to make, feel free to include them. Thank you.

Location in Manual	Comment/Suggestion
_____	_____
	_____
	_____
	_____
_____	_____
	_____
	_____
	_____
_____	_____
	_____
	_____
	_____

CUT ALONG LINE

NO POSTAGE NECESSARY IF MAILED IN U.S.A.
FOLD ON TWO LINES (LOCATED ON REVERSE SIDE), TAPE AND MAIL

FOLD

FIRST CLASS
PERMIT NO. 7284
DALLAS, TEXAS

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

TEXAS INSTRUMENTS INCORPORATED
DIGITAL SYSTEMS DIVISION

P.O. BOX 2909 · AUSTIN, TEXAS 78769

ATTN: TECHNICAL PUBLICATIONS
MS 2146

FOLD

TEXAS INSTRUMENTS
INCORPORATED
DIGITAL SYSTEMS DIVISION
POST OFFICE BOX 2909 AUSTIN, TEXAS 78769