TEXAS INSTRUMENTS

Improving Man's Effectiveness Through Electronics

Model 990 Computer
TX990
Operating System Documentation

MANUAL NO. 944776-9701
ORIGINAL ISSUE 1 OCTOBER 1977
INCLUDES
CHANGE 1 . . . . . 15 DECEMBER 1977
CHANGE 2 . . . . . 1 SEPTEMBER 1978

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INSERT LATEST CHANGED PAGES DESTROY SUPERSEDED PAGES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 Documentation (944776-9701)

Original Issue . . . . . .1 October 1977
Change 1 . . . . . .15 December 1977 (ECN 419813)
Change 2 . . . . . .1 September 1978 (ECN 003580)

Total number of pages in this publication is 116 consisting of the following:

PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.	PAGE NO.	CHANGE NO.
Cover	2	5-9 - 5-10	2		
Eff. Pages	2	5-11 - 5-14B	2		
iii - iv	0	5-15 - 5-16B	2		
v - viii	2	5-17 - 5-18B	2		
1-1 - 1-2	2	5-19 - 5-20H	2		
2-1 - 2-6D	2	5-21 - 5-22	0		
3-1 - 3-8	1	6-1 - 6-10B	2		
3-9 - 3-10	2	Alphabetical Index Div	0		
4-1 - 4-4	0	Index-1 - Index-6	2		
4-5 - 4-8B	2	User's Response	2		
4-9 - 4-10B	1	Business Reply	0		
4-11 - 4-14	0	Cover Blank	0		
5-1 - 5-6	1	Cover	0		
5-7 - 5-8	0				

PREFACE

This manual is directed to the user who intends to modify the TX990 operating system. It is assumed the user is experienced in 990 assembly language and has access to the source code for the TX990 operating system. This manual describes the internal data structures and organization of the operating system. The following is an abstract for each section of this manual.

I Introduction – Describes the scope of the TX990 operating system.

II TX990 Structure and Control – Describes the internal structure and control flow of the TX990 operating system. It is necessary to understand the control flow before modifying the system, so that the impact of any modifications on the system is predictable.

III Privileged Supervisor Call Blocks – Describes the privileged supervisor call blocks. This section also discusses the purpose of each supervisor call, and how it is used.

IV Modifying TX990 – Explains some of the more common reasons for modifying the system. Describes the considerations that must be taken into account before or during the modification of the system. Explains in detail how to modify the operator communication package (OCP), write device service routines (DSR), write extended operations (XOP) and supervisor calls.

V Data Structures – Describes each data structure and the relationship between the data structure and the operating system. This section explains how the system uses the data structure to control system functions.

VI Module Description – Explains the purpose and function of each module so that a user can make intelligent choices on component parts of the TX990 operating system. Each module has a brief functional description and lists other modules that must be included in order to utilize the module effectively.

The following documents contain information referenced in this manual:

Title	Part Number
Model 990 Computer TX990 Operating System Programmer's Guide (Release 2)	946259-9701
Model 990 Computer Assembly Language Programmer's Guide	943441-9701

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION I

SCOPE OF THE TX990 OPERATING SYSTEM

1.1 INTRODUCTION

The purpose of the TX990 operating system is to provide task scheduling functions, memory management, interactive debugging aids, interactive operator control, and a basic disc file management package. The TX990 operating system is designed to be upwardly compatible at the task level with the RX990 and DX10 systems. TX990 is modularly designed so that modules may be selected according to the needs of the user to customize the TX990 operating system. TX990 is a multitasking, single user operating system. Many tasks may execute simultaneously; but in the single dynamic task environment, there is no memory resource scheduling within the operating system. The user must be aware of which tasks require the dynamic memory area, and must execute only one of these tasks at a time. Memory resource scheduling is provided when the multiple dynamic tasks option is selected. Since there is no resource scheduling of certain hardware devices, such as the line printer and card reader, conflicting requests for these devices must be resolved by the user by either generating these devices in record mode during system generation, or by scheduling the tasks so they do not conflict on resource requests. The user is also responsible for preventing a task from inadvertently altering memory outside of its address space and destroying other tasks or the operating system.

SECTION II

TX990 STRUCTURE

2.1 TX990 OPERATING SYSTEM CONTROL FLOW

Control flow through the operating system begins and ends with the task. When a task is bid, the task scheduler places the task on the active task queue according to the task's priority. When the task is the highest priority task on the queue, the task scheduler begins the execution of the task. If the task desires a particular service of the operating system, a supervisor call (SVC) is issued. The supervisor call interface routine decodes the SVC code, the appropriate SVC routine is selected, and control is transferred to it. Upon completion of the SVC or the discovery of an error, control is transferred to a common exit routine. The common exit routine will reactivate the task immediately, end the task's time slice and put the task at the end of the appropriate priority's active task queue, or suspend the task, depending upon the particular SVC.

If an I/O operation or a device generates an interrupt for any reason, control is transferred to the appropriate interrupt handler, and from there to the common exit routine. The scheduler then gives the CPU resource to the highest priority task on the active task queue.

If a fatal error occurs during the processing of an SVC or while a task is executing, control is transferred to the end action routine supplied by the task. If the task does not have an end action routine, the diagnostic task (DTASK) will be executed. DTASK displays an error message and performs the necessary housekeeping functions required to terminate the task. Control is then transferred to the task scheduler where the task is terminated. See figure 2-1.

2.2 TASK SCHEDULER

The task scheduler uses a priority scheme with 131 levels and maintains a list of active tasks by priority level. A task is added to the active task list in each of the following cases:

- When the task is placed in execution (bid).
- When the task is reactivated by another task.
- At the completion of a time slice (if time slicing was selected during SYSGEN).
- At the completion of an SVC that caused the task to be suspended.

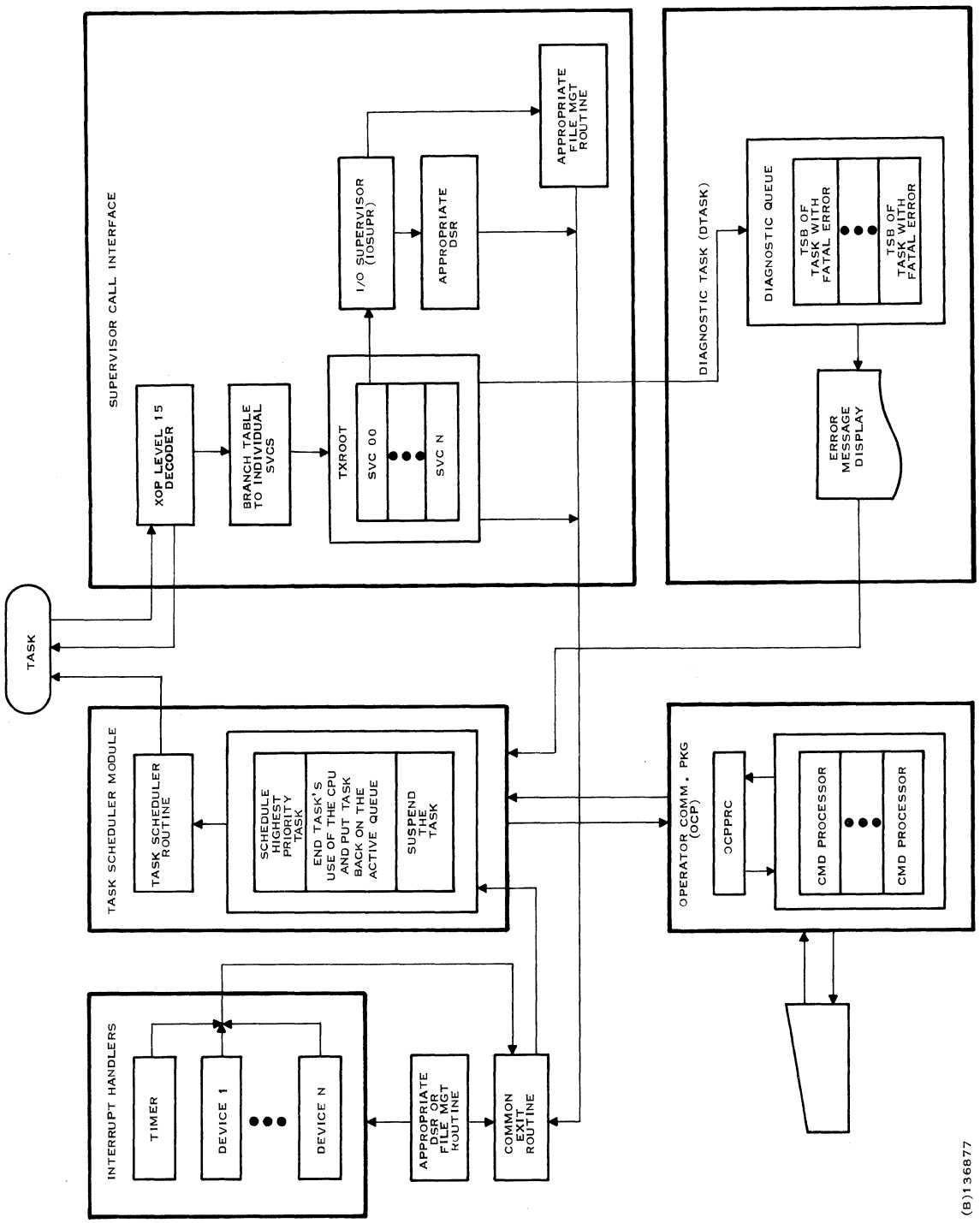


Figure 2-1. TX990 Operating System Control Flow

Figure 2-1. TX990 Operating System Control Flow

The task priorities are defined as follows:

Priority	Description	Priority Ranking
0	Reserved for those system tasks whose functions are necessary to the system's operation.	Highest
Real-Time Priorities 1-127	Designed for real-time tasks which perform monitoring and control of processes.	
Priorities 1, 2, and 3	These are the computation, I/O, and data processing priority levels. Most user applications tasks and the remainder of the system function tasks are installed at these levels. System services tasks should be installed at level 1 (highest), while CPU bound processing tasks should be installed at priority level 3 (lowest).	Lowest

The scheduler time shares the CPU resource between tasks which are installed at the same priority, provided time slicing was selected during system generation. 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 SVC to suspend the task.
- The system suspends the task to await completion of an I/O operation.
- The system suspends the task due to an event which causes a higher priority task to become active. The higher priority task then begins execution. This action is called preemption.

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 highest priority task on the active list.

If time slicing was not selected at SYSGEN time, then the task runs until it is preempted or suspended. Without the time slicing option, tasks installed at equal priority are not preempted by the system for the purpose of sharing the CPU.

To prevent a CPU bound task from completely locking out lower priority tasks (for example, an undebugged task which gets into an infinite loop), the Task Sentry feature has been implemented in TX990. The feature is optional at SYSGEN time. If the Task Sentry is selected, any task which uses the CPU for more than a specified amount of time will have its priority lowered by one level. If the task continues to run without suspending itself and the time is exceeded, the priority is again decremented until the task's priority will go down to the lowest priority level. The amount of time a task is allowed before the priority is lowered is also selectable at SYSGEN time.

2.3 MEMORY MANAGEMENT

User memory is maintained by the memory management routine (module MEMSVC for a single dynamic task or DMEMSVC for multiple dynamic tasks). For the single dynamic task system, memory is not actually allocated. A pointer to the available memory block, immediately following the dynamic task, is returned to the task requesting memory. All tasks requesting memory will be returned this same pointer. For multiple dynamic tasks, the user memory is actually allocated to the requesting task. Each task requesting memory will receive a different block of memory. This memory is returned to the available memory pool when the task terminates or issues Return Memory SVC.

Buffer manager (module TBUFMG) is used to service the system requests for buffers contained in TXDATA (i.e., defined during system generation. See paragraph 5.1 for a description of TXDATA.). The buffer manager has queues of different sizes of memory blocks. These blocks of memory are used as buffers for LUNO assignments, blocking buffers, intertask communications. messages, communication package buffers, etc.

2.4 SUPERVISOR CALL INTERFACE

A supervisor call is the method by which the user task requests that a particular service be performed by the operating system. TX990 implements supervisor calls by using the extended operation (XOP) level 15. The hardware decodes the XOP level as an index into a branch table (defined during system generation and initialized during the loading procedure) that contains the workspace and entry address of the XOP processor. All level 15 XOPs are channeled to a processor in module TXROOT. TXROOT decodes the XOPs operation code, which is the first byte of the supervisor call block, and branches to the appropriate supervisor call processor. TXROOT also has a common return routine for all the supervisor call processors which provides three return entry points that either reactivate the task immediately, end the task's time slice and put the task on the active queue, or suspend the task.

2.5 INPUT/OUTPUT OPERATIONS

All XOP level 15 supervisor calls with a supervisor call code of 00_{16} are handled by module IOSUPR. The I/O supervisor call processor uses the LUNO number obtained from the supervisor call block to determine whether the I/O is directed to a device or a disc file. When the I/O is directed to a device, the appropriate device service routine is called. When the device service routine completes the operation, a routine in IOSUPR, ENDREC, is called to perform system housekeeping. Control is returned to the calling task through the common return routine in TXROOT. When the I/O is directed to a disc file, the calling task's Task Status Block (TSB) is put on a file management queue and the appropriate file management task is bid. The file management task removes the TSB from the queue, performs the I/O service, and places the TSB back on the active queue. If it has no other queued services to resolve, the file management task terminates.

2.6 FILE MANAGEMENT TASKS (FMP1, FMP2, FMP3, FMP4, FUR, VOLUME)

File management tasks are bid by module IOSUPR when an I/O operation is directed to a disc file. A file management task removes TSBs requests from a queue and services the I/O requests. It continues to service the queued requests until the queue is empty. At that time, the file management

task terminates. There are four file I/O tasks: FMP1, FMP2, FMP3, and FMP4 (Task IDs $F0_{16}$, $F1_{16}$, $F2_{16}$, $F3_{16}$). There must be a file I/O task for each disc drive in the hardware configuration for which file support is desired. The fifth task is a file utility task, FUR (Task B_{16}). The file utility task creates, deletes, compresses, and changes the names of files. The sixth task, VOLUME (Task C_{16}), locates the volume and associates with it a diskette drive and the file management task for that drive.

2.7 OPERATOR CONTROL, TASK $0F_{16}$

TX990 has an operator communications package (OCP) that allows the user to interface with the operating system to control, load and execute tasks, manipulate LUNOs, and debug. OCP is an optional task (Task F_{16}) that must be linked to the system so it can access system data structures. OCP is organized so that the user may easily write additional OCP commands for a customized TX990 operating system. (Section IV describes this process.) The module OCPPRC is the control module for OCP commands. It decodes the call and branches to a command

processor. When the command processor has completed its task, it branches to a common return point in OCPPRC. The command processors are packaged in discrete modules so that the user may select only those commands necessary for his dedicated application using a customized TX990 operating system.

2.8 DIAGNOSTIC TASK (DTASK), TASK $0D_{16}$

The diagnostic task is an optional, but highly recommended, linked-in TX990 system task. It has a task I.D. of $0D_{16}$ and is activated by error detection logic in the TXROOT module.

When a task commits a fatal error and its end-action word is less than 16_{10}, the TXROOT error logic removes the task status block from its active queue and puts it on the diagnostic queue, while changing the task's state to "On Diagnostic Queue". After this has been done, the diagnostic task is bid.

DTASK removes a queued TSB and prints the task I.D., error code, and the workspace pointer (WP), status register (ST), and program counter (PC) of the task when the error occurred. The printed PC value usually points to the instruction following the one that caused the error. After the message has been printed, the diagnostic task sets the task status block's kill task flag and returns the task to its active queue to be killed by the scheduler. Before termination, DTASK checks for more entries on its queue.

In a TX990 system with DTASK, a task committing a fatal error causes a message to be printed on LUNO 0. A TX990 system without DTASK places an error task on the diagnostic queue and takes no other action. In such a system, a task that is terminated with a fatal error remains on the diagnostic queue, in the "On Diagnostic Queue" state. Should the diagnostic task commit a fatal error (a system malfunction), the message "HELP" is printed on the system console.

2.9 INITIAL START TASK (STASK), TASK 10_{16}

The TX990 startup task is a linked-in system task that runs only at initial system startup. Its purpose is to demonstrate that the system is up and running. STASK prints the memory size of the machine and the size of the dynamic task area on the system console. The startup task is normally linked after the TXEND module, physically placing it in the dynamic task area. It is overlaid by any user software that is loaded, and therefore does not add to the size of TX990. For this reason, it is recommended that STASK be included in all TX990 systems, except dedicated application systems.

In the normal TX990 configuration, STASK has a task I.D. of 10_{16} and executes only on the initial start of the operating system. If it is desired that STASK be executed on a manual restart of TX990, it is necessary for it to be linked before TXEND and for a task I.D. other than 10_{16} to be assigned to it during system generation.

2.10 SINGLE DYNAMIC TASK LOADER ROUTINE

The loader routine loads the object code, from the file or device assigned to LUNO 2, into the dynamic task area. The task that is loaded has the task I.D. of 10_{16}. A user task that is to use the loader routine must be linked to the operating system, since the object program is loaded into the dynamic task area. The user task must reference three labels: LDRDAT, LDRFLG, and LOADER. LDRFLG is a lock-out flag (see Section V). When the flag is set, only the user who set it may use the task loader. Once the flag is set, the user task must initialize a parameter

block which begins at the label LDRDAT. LDRDAT is a block of memory in the task loader itself and has the following format.

BYTE 0	ADDRESS OF BUFFER	
BYTE 2	PRIORITY LEVEL	PRIVILEGE FLAG

LDRDAT is a two-word parameter block. Bytes 0 and 1 contain the address of an 80-character buffer in the user's task to be used for input by the loader. Byte 2 contains the priority level at which the task will run (See note below). Byte 3 contains the privilege flag: 0 indicates the task is to be loaded as nonprivileged, 80_{16} indicates the task is to be loaded as privileged. Once the TSKLDR parameter block, LDRDAT, is initialized by the user task, the user task must branch to the loader entry point, which is called "LOADER" (via a BLWP instruction). When the task loader completes loading the file to which LUNO 2 is assigned, it returns to the user task. The task loader returns an error code in the left byte of the calling task's register 0.

The module IMGLDR must be included to load a task from a program image file. If the program file being loaded has overlays, the task loader assigns LUNO >10 to the program file. This is used by the overlay loader routine to load overlays. See the link editor manual for details on automatic overlay loading.

The following coding example illustrates the use of the task loader routine. LUNO 2 is already assigned to an object file.

```
        REF     LDRDAT
        REF     LDRFLG
        REF     LOADER

WAIT    DATA    >200,40         TIME DELAY SVC BLOCK
*
BUF     BSS     80              80-CHARACTER BUFFER
*
L1      ABS     @LDRFLG         SET THE SEMAPHORE
        JLT     L2              IF SUCCESSFUL GO SET UP LDRDAT
        XOP     @WAIT,15          ELSE WAIT AND TRY AGAIN
        JMP     L1
L2      LI      R2,LDRDAT       INITIALIZE THE PARAMETER BLOCK
        LI      R1,BUFF
        MOV     R1,*R2+         PUT THE BUFFER ADDRESS IN BYTE 0 OF
*                                 LDRDAT
        LI      R1,>0380        SET THE PRIORITY TO LEVEL 3,
        MOV     R1,*R2            AND MAKE THE TASK PRIVILEGED
*
        BLWP    @LOADER         CALL THE LDR ROUTINE
        MOVB    R0,R0           CHECK THE ERROR CODE
        JNE     ERROR             IF ERROR THEN EXIT
          .
          .
          .
```

NOTE: Priorities are represented as follows:
Priority 0: >00
Real-time Priorities 1-127: >81 to >FF
Priority 1-3: >01 to >03

2.11 MULTIPLE DYNAMIC TASK AND PROCEDURE LOADER

TX990 optionally supports the installation of multiple dynamic tasks and procedures at execution time. These tasks and procedures are loaded by system routines callable by programs linked with the operating system. Dynamic task >10 can still be loaded by the same interface as described in paragraph 2.10. System routines are provided to install task, install procedure, delete task, and delete procedure.

The modules DYNTSK, DTSKLDR (replaces TSKLDR), and DMEMSVC (replaces MEMSVC) must be included to support these routines. Since these routines may not be executed concurrently, the linked-in user program must test and set the loader lock-out flag, LDRFLG, to gain exclusive access to the loader routines. LDRFLG is reset by the TX990 System when the load or delete operation is complete. The following illustrates the calling sequence:

```
LOOP    ABS     @LDRFLG         TEST AND SET FLAG
        JLT     CALL            IF BUSY
        SVC     @DELAY                THEN DELAY
        JMP     LOOP                     AND RETRY
CALL    FQU     $               ELSE
        BLWP    @ITASK                CALL TASK LOADER
        DATA    ITBLK                 CALL BLOCK
```

The calling program must pass a call block, much like a supervisor call block, to the system routine. The format of each block is described in the following paragraphs.

2.11.1 INSTALL TASK CALL BLOCK. The install task routine requires the following data block:

BYTE		
0	0	ERROR CODE
2	LUNO	TASK ID
4	FLAGS	PRIORITY
6	PROC1 ID	0
8	EXTRA MEMORY	

The install task call block data is:

Byte 0 – Not used.

Byte 1 – Error code returned by the system.

Byte 2 – LUNO assigned to input file.

Byte 3 – Installed task ID.

Byte 4 – Flags.

Bit 0 Privileged task.

Bit 1 Do not rewind LUNO.

Byte 5 – Task priority.

Priority 0 = >00
Real-time Priority 1-127 = >81 to >FF
Priority 1-3 = >01 to >03

Byte 6 – Procedure ID.

Byte 7 – Reserved (must be zero).

Bytes 8 - 9 – Allocate extra memory for task.

The LUNO must already be assigned to the file to be loaded. Errors are returned for duplicate task ID, insufficient memory, and nonexistent procedure.

2.11.2 **INSTALL PROCEDURE CALL BLOCK.** The format of the install procedure block is as follows:

BYTE		
0	0	ERROR CODE
2	LUNO	ID

The install procedure call block data is:

Byte 0 – Not used.

Byte 1 – Error code returned by the processor.

Byte 2 – Input LUNO.

Byte 3 – Procedure ID.

The LUNO must already be assigned to the file to be loaded. Errors are returned for duplicate procedure ID and insufficient memory.

2.11.3 **DELETE TASK CALL BLOCK.** The format of the delete task data block is as follows:

BYTE		
0	0	ERROR CODE
2	0	TASK ID

The delete task call block data is:

Byte 0 – Not used.

Byte 1 – Error code returned by the SVC processor.

Byte 2 – Reserved (must be zero).

Byte 3 – ID of task to be deleted.

An error is returned if the task is not terminated.

2.11.4 **DELETE PROCEDURE CALL BLOCK.** The format of the delete procedure data block is the same as for the delete task data block, except ID refers to procedure ID. An error is returned if the procedure is attached to a task.

2.12 CONTROL PROGRAM (CNTROL), TASK 16_{16}

The control program is the control module for the TXDS utility programs. It decodes the user's entries for program name, I/O specifications, and options. It then loads and branches to the specified utility program. When the utility program terminates, the utility executes an end program supervisor call (16_{16}) which activates the rebid task.

2.13 REBID TASK

The system rebid task is the task that is activated whenever an end program supervisor call is issued. For standard TX990/TXDS systems, the TXDS control program is defined as the rebid task. The control program is defined to have the task I.D. of 16_{16}. The user may wish to designate another task as the rebid task. This can be accomplished in the following manner.

1. Place the following statements in the new rebid task.

```
        DEF     REBID
REBID   EQU     ID*>100
```

   Where ID = The task I.D. of the new rebid task.

2. When the TXDS control program is included in the system, link the object code for the new rebid task before the module CNTROL.

SECTION III

PRIVILEGED SUPERVISOR CALLS

3.1 GENERAL

TX990 contains a number of supervisor calls that can only be made by tasks which are privileged. This insures that the user program cannot easily gain access to internal operating system structures and inadvertently modify them.

3.2 GET SYSTEM TABLE

The 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. See Section V for a description of the system table. This supervisor call should be used only by system tasks that require access to internal data structures. A program using this SVC will not be compatible with any other 990 operating system.

3.3 DIRECT DISC I/O

3.3.1 INTRODUCTION. To maintain disc file integrity and security, and to control disc allocation, direct disc I/O is reserved for use by TX990 system tasks. System tasks are those which are executed in the privileged mode.

I/O to disc is controlled by an extended supervisor call block (SCB) as shown in figure 3-1. See the *Model 990 Computer TX990 Operating System Programmer's Guide (Release 2),* number 946259-9701, for a description of the standard supervisor call block. As with any other device, access is through a LUNO assigned to the disc. TX990 has a reserved system (nonreleasable) LUNO (starting at $F0_{16}$) for use by system tasks, such as file management and file utility, for each disc unit configured in the system. Action taken on the I/O opcodes is described in table 3-1. Discs are classified as record-oriented devices by TX990 and need not be opened or closed after or before I/O operations.

0	0	ERROR CODE	
2	I/O OPCODE	LUNO	
4	SYSTEM FLAGS	USER FLAGS	
6	BUFFER ADDRESS		0
8	RECORD LENGTH		0
10	CHARACTER COUNT		0
12	TRACK ADDRESS		
14	SECTORS PER RECORD	SECTOR ADDRESS	

SYSTEM FLAGS
- 0 - BUSY
- 1 - ERROR
- 2-7 - NOT USED

USER FLAGS
- 0 - INITIATE I/O
- 1-5 - NOT USED
- 6 - SECTOR LENGTH SPECIFIED
- 7 - ALLOCATION UNIT I/O

(A)136878

Figure 3-1. Direct Disc I/O SCB

Table 3-1. Direct Disc I/O Opcodes

Opcode	Function	Action
0_{16}	Open	Ignored (device type returned)
1_{16}	Close	Ignored
2_{16}	Close EOF	Ignored
3_{16}	Open Rewind	Ignored (device type returned)
4_{16}	Close Unload	Ignored
5_{16}	Read Format	Return disc/track format
6_{16}	Forward Space	Ignored
7_{16}	Back Space	Ignored
8_{16}	Write Format	Formats track
9_{16}	Read ASCII	Allocation-based read
A_{16}	Read Direct	Track-based read
B_{16}	Write ASCII	Allocation-based write
C_{16}	Write Direct	Track-based write
D_{16}	Write EOF	Ignored
E_{16}	Rewind	Ignored
F_{16}	Read Format	Return disc/track format
10_{16}	Write Deleted Sector	Write deleted code on sector

Before tracks on a disc can be used for data storage using file management, they must be formatted. This may be done by a utility program called :BACKUP/SYS.

Track numbering runs sequentially from zero to the maximum track. Physical read and write operations must start at sector boundaries, but need not include an entire sector, and may cross sector boundaries.

Records must be read and written from the beginning but need not be read or written to the end. Short records written to a disc cause the remainder of the physical record to be zero filled. Sectors are numbered from zero on each physical track.

3.3.1.1 **Direct Disc Call Block.** The direct disc supervisor call block (SCB) is sixteen bytes long, as shown in figure 3-1. The SCB has the following format:

Byte 0 – I/O SVC code.

Byte 1 – Error code returned by TX990.

Byte 2 – I/O OPCODE contains the code of the operation to be performed.

Byte 3 – LUNO. The logical unit assigned to the disc.

Byte 4 – SYSTEM FLAGS. System status in an I/O operation.

Bit 0 — Busy. Set to one when operation is initiated and to zero upon completion.

Bit 1 — Error. Set to one when operation terminates in error. Byte 1 contains the error code.

Byte 5 – USER FLAGS. Set by user prior to operation.

Bit 0 — Initiate I/O. User sets to one for an Initiate Call. When zero, task is suspended until I/O is complete. When one, system initiates operation and returns control to the task.

Bits 1-5 — Not used.

Bit 6 — Sector Length Specified. Set to one if user wants a different sector size. The size, in bytes, is in byte 14 of the SCB. Also set to one to indicate that a logical track I. D. is specified (in byte 14 of the SCB) on a write format operation.

Bit 7 — Allocation Unit I/O. Set to one for allocation unit I/O. Set to zero for physical track I/O.

Bytes 6-7 – BUFFER ADDRESS. Address of data buffer. Aligned on a word boundary.

Bytes 8-9 – RECORD LENGTH. Maximum number of characters to be entered.

Bytes 10-11 – CHARACTER COUNT. For input operations the number of characters actually input. For output operations, the number of characters written.

Bytes 12-13 – TRACK ADDRESS. The physical track address on the disc for track I/O. The allocation unit address for allocation unit I/O.

Byte 14 – SECTORS PER RECORD. This field is used in several different ways. On format track it is the logical track I. D., if user flag bit 6 is set, and a one (1) is returned. On track-based read or write, it is a physical sector length in bytes.

Byte 15 – SECTOR ADDRESS. The sector within a physical track or an allocation unit.

3.3.2 TRACK-BASED I/O

3.3.2.1 General. Track-based I/O is specified when bit 7 of the SCB user flag byte (byte 5) is set to zero. This mode of I/O is used to access the disc in its physical configuration of tracks and sectors. The commands are treated as described below.

3.3.2.2 Read Format (5_{16} and F_{16}). This command requires at least 5 words of buffer space. SCB record length, character count, sectors per record, track address, and sector number are ignored. Ten bytes of track-based disc and track-based data are returned in the user buffer as shown in figure 3-2. The track-based read format data is:

Bytes 0-1 – WORDS/TRACK. Number of unformatted words on each track of the disc.

Byte 2 – SECTORS/TRACK. Number of sectors on each track of the disc.

Byte 3 – OVERHEAD/RECORD. Number of words required for each formatted record on a track.

Byte 4 – # OF HEADS. Number of heads addressable on the disc unit.

Bits 0-4

Bytes 4-5 – # OF CYLINDERS. Number of cylinders addressable on the disc units.

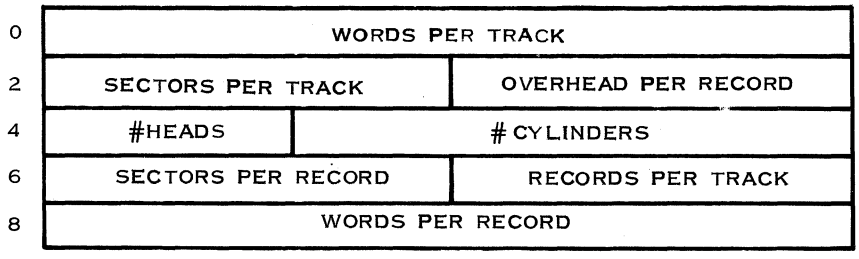
Bits 5-15

Byte 6 – SECTORS/RECORD. Number of sectors spanned by each record on the track.

Byte 7 – RECORDS/TRACK. Number of records the track is formatted into.

Bytes 8-9 – WORDS/RECORD. Length of records on track (in words).

3.3.2.3 Write Format (8_{16}). This command requires an SCB record length and track address. It formats the addressed track to the given record length and returns the resulting number of sectors per record in byte 14 of the SCB. User buffer, character count, and sector number are ignored.



(A)136879A

Figure 3-2. Track-Based Read Format Data

3.3.2.4 **Read Direct (A_{16}), Write Direct (C_{16}).** These commands transfer data to/from the user buffer by standard TX990 conventions. The SCB record length determines the maximum number of characters received during a read and the SCB character count determines the number of characters transmitted during a write. No distinction is made between ASCII and direct I/O transfers, all are binary data. Data transfers can span records and tracks.

Data is always transferred on a word basis, that is, buffer addresses and byte counts are truncated to even numbers. Note that the complete disc address of track number, sector number, and track format (sectors per record) is required for data transfers.

3.3.3 ALLOCATION UNIT-BASED I/O

3.3.3.1 **General.** Allocation unit-based I/O is specified when bit 7 of the SCB user flag byte is set to one. This mode of operation is provided for the convenience of the TX990 file management package. Special applications programs that do direct disc I/O usually use track-based operations.

When operating in the allocation unit mode, the DSR restructures the physical layout of the disc to a format more easily used by TX990 file management. This capability allows the file management system to be implemented without any prior knowledge of the characteristics of the device to be used for file storage, other than the fact that it is a random-access, mass storage device.

With track-based I/O, the disc is addressed in terms of track and sector. For allocation unit operations, the disc is addressed in terms of allocation unit and physical record.

The AU is the basic unit of file space allocation used by file management. An allocation unit is a set of 6 physical records; i.e., sectors, located on one or two tracks of a diskette. The 6 records composing a single AU are NOT CONTIGUOUS (table 3-2). Thirteen AUs fill 3 tracks on the diskette. Table 3-2 lists the addresses of each of the 13 AUs on tracks 0, 1, and 2. The sequence of sectors repeats throughout the remaining tracks on the diskette. Table 3-3 is a sampling of initial records for other AUs on the remaining tracks.

3.3.3.2 **Read Format (5_{16} and F_{16}).** The Read Format command returns sixteen bytes of information to the calling program's buffer, as shown in figure 3-3. A disc to be used with TX990 is always formatted to the number of sectors per record returned by this call. The allocation unit-based read format data is:

Bytes 0-1 – BYTES PER RECORD. The number of bytes per physical record on the disc.

Bytes 2-3 – FCB SIZE. The number of bytes needed to contain the File Control Block (FCB).

Bytes 4-5 – RECORDS PER ALLOCATION UNIT. The number of physical records in each allocation unit.

Bytes 6-7 – MAXIMUM NUMBER OF ALLOCATION UNITS. The number of allocation units available on the disc.

Bytes 8-11 – DIRECTORY START AND SIZE. The allocation unit where the directory begins (2 bytes) and its length in physical records (2 bytes). The directory always starts at physical record 0 of its allocation unit.

Bytes 12-15 – ALLOCATION TABLE START. The allocation unit (2 bytes) and physical record (2 bytes) where the disc allocation table starts.

Bytes 16-17 – BAD ALLOCATION TABLE START. The physical record on the same allocation unit as the Allocation Table, bytes 12-13, where the disc allocation table for bad allocation units starts.

Bytes 18-21 – DEVICE INFORMATION BLOCK START. The allocation unit (2 bytes) and physical record (2 bytes) where the disc device information block is located.

Bytes 22-25 – HASH TABLE START. The allocation unit (2 bytes) and physical record (2 bytes) where the directory hash table begins. This field is currently unused.

Byte 26 – DISC TYPE. A unique code for each type of disc.

Byte 27 – Unused.

Table 3-2. Allocation Unit Record Assignments

AU, Record	Track, Sector	AU, Record	Track, Sector	AU, Record	Track, Sector
0, 0	0, 4	4, 2	1, 4	8, 4	2, 4
0, 1	0, 10	4, 3	1, 10	8, 5	2, 10
0, 2	0, 16	4, 4	1, 16	9, 0	2, 16
0, 3	0, 22	4, 5	1, 22	9, 1	2, 22
0, 4	0, 2	5, 0	1, 2	9, 2	2, 2
0, 5	0, 8	5, 1	1, 8	9, 3	2, 8
1, 0	0, 14	5, 2	1, 14	9, 4	2, 14
1, 1	0, 20	5, 3	1, 20	9, 5	2, 20
1, 2	0, 0	5, 4	1, 0	10, 0	2, 0
1, 3	0, 6	5, 5	1, 6	10, 1	2, 6
1, 4	0, 12	6, 0	1, 12	10, 2	2, 12
1, 5	0, 18	6, 1	1, 18	10, 3	2, 18
2, 0	0, 24	6, 2	1, 24	10, 4	2, 24
2, 1	0, 5	6, 3	1, 5	10, 5	2, 5
2, 2	0, 11	6, 4	1, 11	11, 0	2, 11
2, 3	0, 17	6, 5	1, 17	11, 1	2, 17
2, 4	0, 23	7, 0	1, 23	11, 2	2, 23
2, 5	0, 3	7, 1	1, 3	11, 3	2, 3
3, 0	0, 9	7, 2	1, 9	11, 4	2, 9
3, 1	0, 15	7, 3	1, 15	11, 5	2, 15
3, 2	0, 21	7, 4	1, 21	12, 0	2, 21
3, 3	0, 1	7, 5	1, 1	12, 1	2, 1
3, 4	0, 7	8, 0	1, 7	12, 2	2, 7
3, 5	0, 13	8, 1	1, 13	12, 3	2, 13
4, 0	0, 19	8, 2	1, 19	12, 4	2, 19
4, 1	0, 25	8, 3	1, 25	12, 5	2, 25

Table 3-3. Sampling of AUs on a Diskette

AU	Track, Sector	AU	Track, Sector
13	3,4	182	42,4
26	6,4	195	45,4
39	9,4	208	48,4
52	12,4	221	51,4
65	15,4	234	54,4
78	18,4	247	57,4
91	21,4	260	60,4
104	24,4	273	63,4
117	27,4	286	66,4
130	30,4	299	69,4
143	33,4	312	72,4
156	36,4	325	75,4
169	39,4	333	77,10

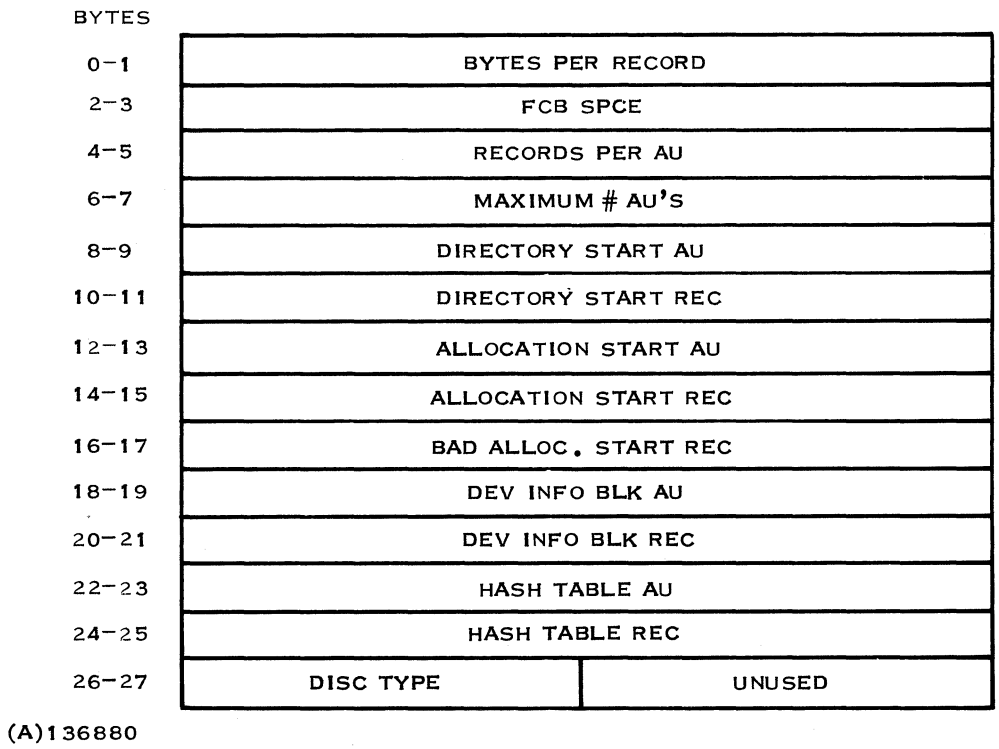


Figure 3-3. Allocation Unit-Based Read Format Data

3.3.3.3 **Write Format (8_{16}).** The Write Format command is not defined for allocation unit-based I/O.

3.3.3.4 **Read ASCII (9_{16}), Write ASCII (B_{16}).** The allocation unit-based read and write operations are identical to the track-based operations previously described provided that all references to "track" are changed to "allocation unit" and all references to "sector" are changed to "physical record".

The number of sectors per record is fixed for allocation unit I/O, therefore this field in the SCB is ignored.

3.3.4 FLOPPY DISC SPECIAL OPERATIONS.

3.3.4.1 **General.** There are certain operations that can be done on the floppy disc that are not generally supported on other discs. These operations should be avoided in any software intended to be transferred to other disc types.

3.3.4.2 **Sector Length Specified.** The floppy disc controller allows the length of a sector to be specified, the allowable range being 2 to 128 bytes. The sector length must be even, and is set to 128 bytes by the DSR unless specifically overridden. To specify the sector length for an I/O operation, bit 6 of the SCB user flags is set to a one and the number of characters per sector is placed in byte 14 of the SCB (sectors/record is always one for the floppy disc and the DSR normally ignores this field). The sector length may be specified only for track-based read and write operations.

3.3.4.3 **Logical Track Specified.** The floppy disc controller allows a track to be formatted with an I.D. other than the physical track the head is positioned over. This capability is necessary to format a diskette that is to be used for data interchange with certain other vendors' equipment.

The DSR always writes the track I.D. to match the physical track unless specifically overridden. To specify the track I.D. to be written, bit 6 of the SCB user flags is set to a one and the track I.D. is placed in byte 14 of the SCB. The track I.D. may be specified for write format operations only.

3.3.4.4 **Write Deleted Sector (10_{16}).** The floppy disc controller can "delete" a sector by writing a special data pattern on it. To delete a sector, I/O opcode 10_{16} is executed with the track and sector address specified. Applies to track-based I/O only.

3.3.4.5 **Floppy Disc Format Restrictions.** The floppy disc can be formatted only in the one sector per record configuration. For this reason, the DSR always ignores this field in the SCB. For software to be transportable to other disc types, it is desirable that this field be filled in correctly for track-based I/O, even though it is not used.

3.3.4.6 **Sector Interleaving.** To more closely match the I/O rate of the floppy disc with the execution rate of software on the 990, the disc sectors are interleaved. This interleaving is in effect only for allocation unit-based operations, and is a simple mapping of logical to physical sectors. The mapping used is shown in table 3-4 and is illustrated in tables 3-2 and 3-3, and is such that four or five logical sectors pass by the read/write head for every disc revolution.

Table 3-4. Floppy Disc Sector Interleaving

Logical	Physical	Logical	Physical
0	4	13	5
1	10	14	11
2	16	15	17
3	22	16	23
4	2	17	3
5	8	18	9
6	14	19	15
7	20	20	21
8	0	21	1
9	6	22	7
10	12	23	13
11	18	24	19
12	24	25	25

3.3.5 **DISC DSR ERRORS.** The DSR can return the following status codes:

Code	Meaning
00	No error
02	Invalid I/O opcode
04	Record lost due to power failure
06	Time-out or abort
11	I. D. error
12	No address mark found
15	Data error
19	Disc not ready

Code	Meaning
1A	Disc write-protected
1B	Unit check
1C	Invalid disc address
1D	Seek error
1E	Deleted sector detected

3.3.6 DISC READ FORMAT DATA (DISKETTE)

3.3.6.1 Track-Based Data.

Words per track:	1664
Sectors per track:	26
Overhead per record:	0
Number of heads:	1
Number of cylinders:	77
Sectors per record:	1
Records per track:	26
Words per record:	64

3.3.6.2 Allocation Unit-Based Data.

Bytes per record:	128
FCB size (bytes):	96
Records per AU:	6
Number of AUs:	333
Directory start/size (AU/rec):	5,6
Allocation start (AU/rec):	4,1
Bad allocation start (rec):	2
Device info blk (AU/rec):	4,0
Hash table start (AU/rec):	0,0
Disc type:	0

3.4 INITIALIZE DATE AND TIME SUPERVISOR CALL $3B_{16}$

The Initialize Date and Time SVC sets the system date and time from values supplied in the buffer referenced by the call block. Only privileged tasks may issue this call. The SVC consists of four bytes, aligned on a word boundary. Byte 0 contains the SVC code, $3B_{16}$. Byte 1 will have an error code of FF_{16} returned if the call is made by a nonprivileged task. Bytes 2 and 3 contain the address

SECTION IV

MODIFYING TX990

4.1 GENERAL

The following sections describe the modifications to the TX990 operating system most commonly made by users. These modifications include adding new XOP routines, adding new SVC processors, adding device service routines (DSR) for nonstandard devices, and adding new processors to the operator communication package (OCP). To make any of these additions, the user must code and assemble modules to support these functions and include the object modules in the system at system generation time. For a detailed discussion about the inclusion of user modules during system generation, see the *TX990 Operating System Programmer's Guide.*

TX990 is designed as a general purpose operating system that can be tailored to fit the user's needs. In some cases, the user may require support from the operating system that is not provided in the standard configuration. This support can be supplied by user-generated XOP routines or SVC processors (XOP 15). Another use for a new XOP or SVC processor is to implement a commonly used subroutine that the user does not wish to link with each program.

TX990 is designed to be a total system, but is constructed in a way that allows the user to build a dedicated system. In doing so, the user may use devices not directly supported by the TX990 operating system. To support these devices, the user writes a DSR that conforms to a set of standards and modifies the standard physical device table (PDT) to support the DSR.

There are a few guidelines that should be followed when modifying the system. Enhancements to the system should be made as discrete modules to maintain the modularity of the operating system. When system tasks are added, they need not be linked with the system itself, but can use the Get System Table supervisor call to obtain the pointers to the system data structures. This allows users to link the task with the memory resident portion of the operating system or to maintain the task on disc. All utilities should be written so that the utility may either be linked with the memory resident portion of the operating system or be disc resident.

4.2 SUPPORT OF NONSTANDARD DEVICES

Before generating code to support a nonstandard device, the user must have thorough understanding of the following information:

1) Device interface – Most devices have a set of common characteristics, but each device usually has its peculiarities which must be dealt with by the programmer. An in-depth understanding of each CRU "Line" in the interface or the operation of the TILINE* is necessary before attempting to write code to control the device. A description of a device interface may be found in a hardware reference manual for the device.

2) Interrupt handling on 990 Family of Computers – The programmer should understand how interrupts are handled on the 990.

3) Basic data structures, program structures, and coding techniques used in writing device service routines.

* TILINE is a registered trademark of Texas Instruments Incorporated.

In order to support a nonstandard device, the user must generate three separate sections of code: A physical device table, an interrupt handler routine, and a device service routine (DSR). The physical device table is created by the system generation program, but the user interactively supplies the generation program with the nonstandard information. The user codes the DSR and the interrupt handler, which are typically contained in the same module, but may be separate modules.

4.2.1 PHYSICAL DEVICE TABLE. Each device in the system that can be accessed by a LUNO has a physical device table (PDT) associated with it. The PDT contains information used by the operating system to support the device. The PDT for a new device is created by the system generation program (GENTX), with the user interactively specifying the new device as a special device (SD). After the standard device requests, CRU base address, access mode, interrupt level, and time-out count, the following inputs are requested by GENTX:

1) CRU interrupt line – The CRU bit displacement from the CRU base address for the bit that is "set" when a device interrupt occurs.

2) Entry label of DSR – The label of the first word of the DSR. An I/O request initiated by a supervisor call is entered two words beyond the label.

3) Entry label of interrupt routine – The label at the entry point to the interrupt routine. The interrupt routine is entered at this point when an interrupt occurs. The interrupt routine processes the interrupt, usually by branching to the appropriate point in the DSR.

4) Interrupt branch label – The initial address of the routine within the DSR to which the interrupt routine branches. The address is placed in register 6 of the DSR workspace when the operating system is initially loaded. The initial label usually points to the routine that handles interrupts when the device is idle.

5) Extension data – Assembler source code that is placed in the extension field of the PDT. This information is totally device dependent, and the field may be empty when the new DSR does not require any extra data or temporary storage words.

4.2.2 INTERRUPT ROUTINE. When a device is interrupt driven, an interrupt routine must be supplied by the user. The functions of this routine are to reset the time-out count (for keyboard devices), reset the device interrupt, and transfer control to the appropriate point within the DSR. The interrupt routine is entered with the DSR workspace of the PDT as the workspace (see figure 4-1). The interrupt routine should perform the following functions:

1) Reset the device time-out count – When the DSR is waiting for an interrupt, the operating system decrements the device time-out count each system-time interval. When the time-out count is depleted before an interrupt occurs, the current I/O operation is aborted. The interrupt routine for keyboard devices must reset the time-out count to its original value. Both the original time-out count and the running count are contained in the device information block of the PDT (figure 4-1).

2) Reset the device interrupt – This function depends on the particular device.

3) Branch to the interrupt entry point – By convention, register 6 of the workspace contains a pointer to the DSR routine to be entered.

PDT STRUCTURE

		SYSGEN
R0	PDT LINK	PDT LABEL
R1	SCB ADDR.+2	0
R2	DEVICE STATUS	NOTE 1
R3	DEVICE CHARACTERISTICS	NOTE 2
R4	DEVICE INFO. BLOCK ADDR.	$+24
R5	SCRATCH OR WS ADDR. (TERM. AND MULTI-UNIT)	0 OR WS LABEL
R6	SCRATCH OR LABEL NAME	0 OR LABEL
R7	SCRATCH	0
R8	SCRATCH	0
R9	SCRATCH	0
R10	SCRATCH	0
R11	SCRATCH	0
R12	CRU OR TILINE	CRU OR TILINE
R13	WP	0
R14	PC	0
R15	ST	0

DEVICE INFO. BLOCK

0	0	TSB ADDRESS	0
2	2	DSR ADDRESS	LABEL
4	4	TIME OUT COUNT	DEV. DEP.
6	6	TIME OUT COUNTER	0
8	8	INTERRUPT ROUTINE ADDRESS	LABEL
10	A	PDT QUEUE POINTER - LAST	0
12	C	PDT QUEUE POINTER - FIRST	0
14	E	LDT POINTER	0
16	10	PDT BUSY INDICATOR	0
18	12	SAVED WP	0
20	14	SAVED ST	0

DSR TEMP STORAGE VARIABLE - DEV. DEP.

NOTE 1

DEVICE STATUS
BIT 0 = NOT USED
1 = NOT USED
2 = INITIATE I/O
3 = KILL TASK
4 = CLOSE DEVICE
5 = RE-ENTER ME
6 = NOT USED
7 = ABORT PRB CALL
8-11 = NOT USED
12-15 = INT. LEVEL -1

NOTE 2

DEVICE CHARACTERISTICS
BIT 0 = FILE ORIENTED DEVICE
1 = TILINE
2 = TIMEOUT DEVICE
3 = PRIVILEGED
4 = CONSOLE
5 = COMMUNICATION DEVICE
6 = DEVICE = 911
7-11 = NOT USED
12-15 = DEVICE TYPE

(A)136881

Figure 4-1. PDT Structure

The following is an example of an interrupt routine:

```
INTRPT   MOV      @4(R4),@6(R4)    Reset device time-out count
           .
           .                       Code to reset device interrupt
           .
           .
         B        *R6              Branch to interrupt entry point
```

Register 4 points to the device information block and bytes 4-5 and 6-7 in the device information block contain the fixed time-out count and decrementing time-out count.

4.2.3 DEVICE SERVICE ROUTINE. The device service routine (DSR) consists of several parts, including code to perform the initialization required when the power is applied, and the code to process an abort or time-out operation. The section of the DSR entered through the entry point processes I/O supervisor calls for the device. When the DSR is interrupt driven, there must be an interrupt routine to process interrupts. Usually, DSRs are written reentrantly so that they may be used by all devices of the same type in the system. In this case, the data area is the PDT and the procedure is the DSR.

The first word of the DSR must contain the address of the entry to the routine that performs the initialization when the power is applied. The second word must contain the address of the entry to the routine that performs abort or time-out functions. The third word is the first word of the portion of the DSR that processes I/O calls. The following is an example of the source code for a DSR:

```
DSR      DATA PWRUP       Address of initialization routine
         DATA ABORT       Address of abort/time-out routine
START    . . . . .        First word of I/O call processor
```

The DSR consists of the following routines:

1) Power-Up/Restart

2) Abort/Time-Out

3) I/O Call Handler

4) Unsolicited Interrupt Handler

5) Interrupt Handler

6) Exit Routine

Figure 4-2 is an example of the typical structure of the DSR.

```
        IDT   'GLBDSR'
        DEF   GLUBER, GLBNCH, GLBINT
        REF   ENDRCD, SETWPS, BZYCHK
* ALSO NEED TO REF ANY OTHER SYSTEM ROUTINE LABELS *
GLUBER DATA GLBPWR            POWER UP ENTRY
        DATA GLBABT           ABORT/TIME-OUT ENTRY
* ENTRY HERE FOR I/O SVC'S
        LIMI 0                MASK INTERRUPTS
          .
          .
          .
        BL    @SETWPS         SYSTEM ROUTINE
       (I/O CALL HANDLER)
        B     @ENDRCD         RETURN TO I/O SUPERVISOR
* UNSOLICITED INTERRUPT HANDLER
GLBNCH EQU  $
          .
          .
          .
        RTWP
* COMMON INTERRUPT HANDLER
GLBINT EQU  $
       (RESET THE INTERRUPT)
        B     *6              GO TO BRANCH LABEL
* POWER UP/RESTART ENTRY
GLBPWR EQU  $
          .
          .
          .
        RTWP
* ABORT / TIME-OUT ENTRY
GLBABT EQU  $
          .
          .
          .
        B     @ENDRCD
        END
(A)136882
```

Figure 4-2. Typical Program Structure of DSR

4.2.3.1 **Power-Up/Restart Routine.** When the operating system is loaded or restarted, either manually or through the power-up interrupt, each DSR in the system is entered at its power-up entry point. This routine should perform the following functions:

1) Reset the interface.

2) Enable device interrupts if the device is interrupt driven.

3) Perform any initializations required by the particular device.

4) Abort in progress I/O – Check to see if I/O is in progress. This can be done with the following instructions:

```
        MOV     R1,R7           Move SCB address +2 to REG 7
        BL      @BZYCHK         System routine to check if PDT busy
        JMP     NOTBSY          Not busy return
        LI      R6, EXIT        Busy, set up exit
        RTWP                    Return
NOTBSY  LI      R6,SPRINT       Set up vector for spurious interrupt
        RTWP                    Return
```

If the device was busy, the error flag and error code (Error Code 4) in the SCB are set by BZYCHK. The DSR should then set up a branch to the normal exit point, which calls the end record processor. When the next interrupt occurs, the interrupt routine branches to the exit routine of the DSR. If the PDT is not busy, the interrupt branch vector should be set to the unsolicited interrupt routine of the DSR. The "RTWP" returns control to the operating system.

4.2.3.2 **Abort/Time-Out Routine.** The operating system enters the DSR at the abort/time-out entry point in response to an abort I/O call or when a device time-out occurs. This routine must perform the following functions:

1) Set error code in SCB – The routine should call the system routine BZYCHK to verify that the PDT is busy. If the PDT is not busy, no more processing is necessary. If the PDT is busy, the error code in the SCB should be set to 6. The error flag has already been set by BZYCHK.

2) Reset the interface – Perform the operations needed to abort the current device I/O.

3) Other clean up – Perform any functions necessary to clean up any flags or temporary information in the PDT.

4) Branch to exit routine.

4.2.3.3 **I/O Call Handler.** The I/O Call Handler is the main part of the DSR. The I/O Call Handler performs the following functions:

1) Decode I/O Opcode – The DSR must decode the opcode in the SCB, determine the I/O operation requested, and transfer control to the appropriate opcode processor.

2) Perform the I/O operation – The processor initiates the desired operation and loads register 6 with the interrupt branch label. When the next interrupt occurs, the interrupt routine branches to the address in register 6. If the processor is waiting for an interrupt to signal the operation complete, the DSR is exited with a "RTWP" returning directly to the operating system. If the operation is completed, the processor must exit the DSR through the exit routine.

4.2.3.4 **Unsolicited Interrupt Handler.** The unsolicited interrupt routine handles interrupts that occur unexpectedly. When the system is initially loaded or the device is idle, register 6, which contains the interrupt branch address, points to this routine. The DSR should set register 6 to point to this routine whenever an operation terminates. This routine should determine the reason for the interrupt and act accordingly.

4.2.3.5 Exit Routine. The Exit Routine should perform the final cleanup of the PDT and/or interface before the DSR exits to the end-record routine via a B @ENDRCD instruction. The interrupt branch vector, register 6, should be set to the unsolicited interrupt routine. The end-record routine closes the device (if record oriented) by clearing the TSB in the PDT, clears the busy flag in SCB and PDT, and reactivates the task (if suspended on I/O).

4.2.3.6 Programming Techniques. When handling I/O calls, the DSR frequently executes a CRU operation and then waits for notification that the operation is complete. If this notification is an interrupt, then the DSR may use the following strategy:

```
LI      R6,ADDR     Set up entry address
RTWP                Return
```

When the interrupt occurs, control will be passed to 'ADDR' in the DSR after the interrupt line has been reset by the interrupt routine.

In some cases, it is necessary to await the occurrence of an event which generates no interrupt. Therefore, the DSR must poll for the event. The user may set the DSR timer reentry flag (bit 5 in R2 of PDT) so that the DSR will be entered at the next system-time interval to test for the event. The code to accomplish this is as follows:

```
LI      R6,ADDR     Set reentry address
ORI     R2,>0400    Set timer reentry flag
RTWP                Return
```

When the DSR is reentered because of the timer, the system resets the flag. If the desired event has not occurred, the flag must be set again by the DSR. If an interrupt occurs, the interrupt routine branches to 'ADDR'.

4.2.3.7 Bidding Tasks from DSR's. The user's application often requires that certain events (usually detected in the interrupt handler part of the DSR) be responded to quickly by a task. Hence, the capability to place a task into execution (or bid a task) from the DSR is provided by the subroutine "BID0". To call BID0, the user must have the task ID in the most significant byte of R1 and the bid parameters in R2 and R3, and then must furnish the following code:

```
BL @BID0
```

The contents of registers R7, R8, R9, and R10 are destroyed. Upon returning from subroutine BID0, the most significant byte of R7 is set to the previous task state if the task's TSB is found. A task state value of zero indicates that the task is active. Otherwise, the MSB of R7 is set to >FF to indicate that the TSB was not found.

4.3 EXTENDED OPERATION ROUTINES

When an extended operation is required, the user must write a routine to perform the required processing. The user supplies the extended operation number (level), the entry point, and the workspace address at system generation time. The system places the entry point and the workspace address in the memory locations corresponding to the extended operation level. When a user task executes an XOP command with that level number as the operand, the computer places the effective address of the command in workspace register 11 of the XOP routine workspace, and performs a context switch to the XOP routine. That is, the XOP routine

workspace becomes the active workspace, XOP (the entry point) is placed in the PC, the previous contents of the WP register are placed in workspace register 13, the previous contents of the PC are placed in workspace register 14, and the previous contents of the ST register are placed in workspace register 15.

The XOP routine may retrieve parameters from the calling program by indirect addressing through register 11 or by using indexed addressing with register 11. For example:

MOV	*R11, R7	Move parameter pointed at by register 11 to register 7.
MOV	@2(R11), R7	Move parameter which is 1 word (2 bytes) past the word pointed at by register 11 to register 7.

The TX990 operating system provides three entry points for returning control to the system from an extended operation routine. When a routine returns control to XOPRT1, the calling task continues execution immediately. When a routine returns control to XOPRT2, the calling task is suspended. When a routine returns control to XOPRT3, the current time slice of the calling task is terminated, and the task is placed on the active list according to its priority just as if it had completed the time slice. The extended operation routine must return control at one of these points, as appropriate. The following examples show these returns:

```
B       @XOPRT1     Return and continue execution of calling task.
B       @XOPRT2     Return and suspend calling task.
B       @XOPRT3     Return, terminating current time slice of
                    calling task.
```

An extended operation routine must externally define (DEF directive) the workspace address and the entry address. It must also externally reference XOPRT1, XOPRT2, or XOPRT3, whichever return point is used by the routine. More than one of these return points may be used, but each must be referenced.

An extended operation may use the workspace without any restriction except that if the contents of workspace registers 13, 14 and 15 are altered, they must be restored before returning control to the system.

4.4 USER-SUPPLIED SUPERVISOR CALL ROUTINES

When a supervisor call (XOP 15) that is not provided by the system is required, the user must write a routine to perform the required processing. The user specifies the call code for the supervisor call and the entry point during execution of GENTX. The user-defined supervisor call codes begin at 80_{16}. Codes 0 through $7F_{16}$ are reserved for system-defined supervisor calls. The system places the entry point in a table from which it is accessed when the call code is recognized in a supervisor call. The system transfers control to the user's routine with the following instruction:

```
B       *R6         R6 contains the entry point of the user's
                    routine.
```

A system workspace having the contents shown in figure 4-3 is the active workspace when the supervisor call routine is entered. The user's routine may destroy the contents of any workspace register except workspace registers 13, 14 and 15. If the user's routine alters the contents of workspace register 13, 14 or 15, the routine must store the contents of the register and restore the contents before returning control to the system. Register 8 contains the address of the Task Status Block (TSB) of the task issuing the supervisor call.

Return to the system utilizes the same entry points defined for extended operation routines, and the same three options apply. The supervisor call routine must branch to XOPRT1, XOPRT2, or XOPRT3, as previously described.

The user must externally define (DEF directive) the entry point and externally reference (REF directive) the return point or points used in the routine.

4.5 OPERATOR COMMAND PROCESSING

Processing of operator commands is performed by five or more modules of the Operator Communication Package (OCP). The OCP consists of four required modules and from one to five optional command processor modules. 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.

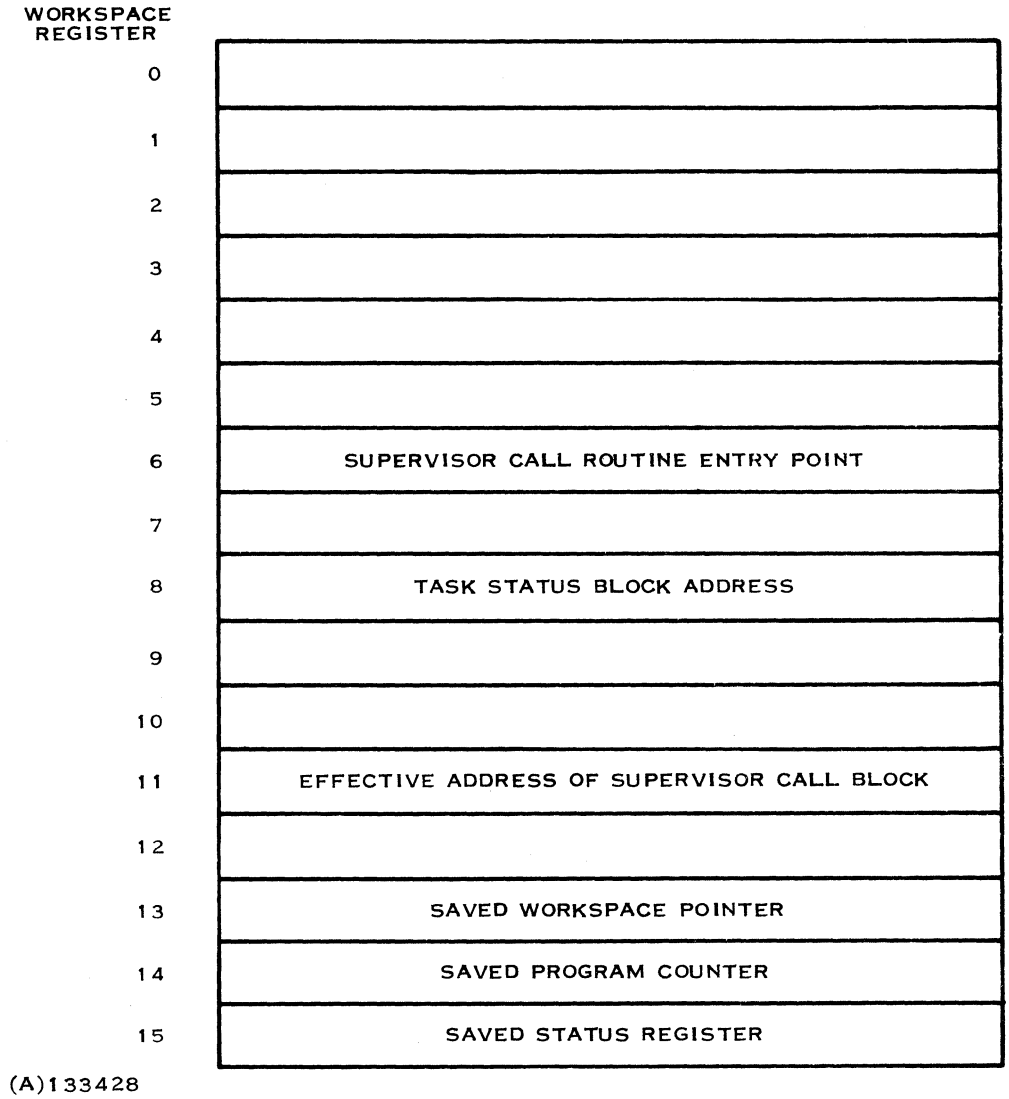


Figure 4-3. Workspace Contents for Supervisor Call Routine

Table 4-1 lists the OCP modules supplied by Texas Instruments. OCP executes as a system task, consisting of a data division (OCPTSK) and a procedure division (OCPPRC). Module OCPTBL contains tables of data that support the command processors that execute as subroutines of the procedure division. Module OCPEND contains external definitions to satisfy external references to the optional modules. It contains a dummy command word table to process optional commands that have been omitted and external definitions to satisfy references to error messages in optional modules. OCPEND must always be linked following the other OCP modules.

Figure 4-4 shows the flow of the processing of OCP commands. After the command line is entered, the pre-scan logic of the OCPPRC module removes embedded blanks and inserts commas for separators where blanks were used. When the last command in the command line has been processed, control returns to the logic that enters a new command line. Otherwise, OCP decodes the command, branching to print an error message when the command code is not in the command tables. Decoding the command results in passing control to the appropriate command processor. After performing the command processing, the command processor returns control to the OCPPRC module; those commands that perform repetitive I/O reenter the command processor. Others return control to test for additional commands in the command line.

Table 4-1. OCP Modules

Name	Description	Linking Order
OCPTSK	OCP Data Division	First
OCPTBL	OCP Tables	Second
OCPPRC	OCP Procedure Division	Third
OCPLRT	Command Processors for ALUNO, RLUNO LPROG, and EXECUTE commands.	Between OCPPRC and OCPEND
DOCPLRT	Command Processors for ALUNO, RLUNO, LPROG, EXECUTE, ITASK, IPROC, DTASK, and DPROC commands. (Replaces OCPLRT in multiple dynamic task system.)	Between OCPPRC and OCPEND
OCPSLD	Command Processors for LMEM, DMEM, SBKPT, CBKPT, ADD, SUB, and JMP commands.	Between OCPPRC and OCPEND
OCPIOU	Command Processors for STASK, SIO, REWIND FSPACE, and BSPACE commands.	Between OCPPRC and OCPEND
DOCPIOU	Command Processors for STASK, SIO, REWIND FSPACE, BSPACE, and SPROC commands. (Replaces OCPIOU in multiple dynamic task system.)	Between OCPPRC and OCPEND.
OCPTAD	Command Processors for TIME and IDATE commands.	Between OCPPRC and OCPEND
OCPTLD	Command Processors for DWKSP, KIO, and KTASK commands.	Between OCPPRC and OCPEND
OCPEND	Dummy external definition module.	Last

Decoding of a command involves a table in the OCPTBL module and tables in the applicable command processor module. Each command processor module contains a Command Word Table having a two-word entry for each command processor in the module, and a terminator. The first word of the entry contains the two-character command code; the second word contains the entry point of the command processor for the command. The OCPTBL module contains a Command Word Table Address Table, consisting of the addresses of the Command Word Tables in each of the command processor modules linked to OCP.

Upon entry to a command processor, workspace registers 1, 6, 10, and 15 contain the following information:

- Register 1 contains the two-character command code.
- Register 6 contains the address of module OCPTSK.

- Register 10 contains the address of the separator (either a comma or a period) at the end of the command word. If the command has one or more operands, the first operand begins at the next address.

- Register 15 contains the address of subroutine GETHEX. (Described in a subsequent paragraph.)

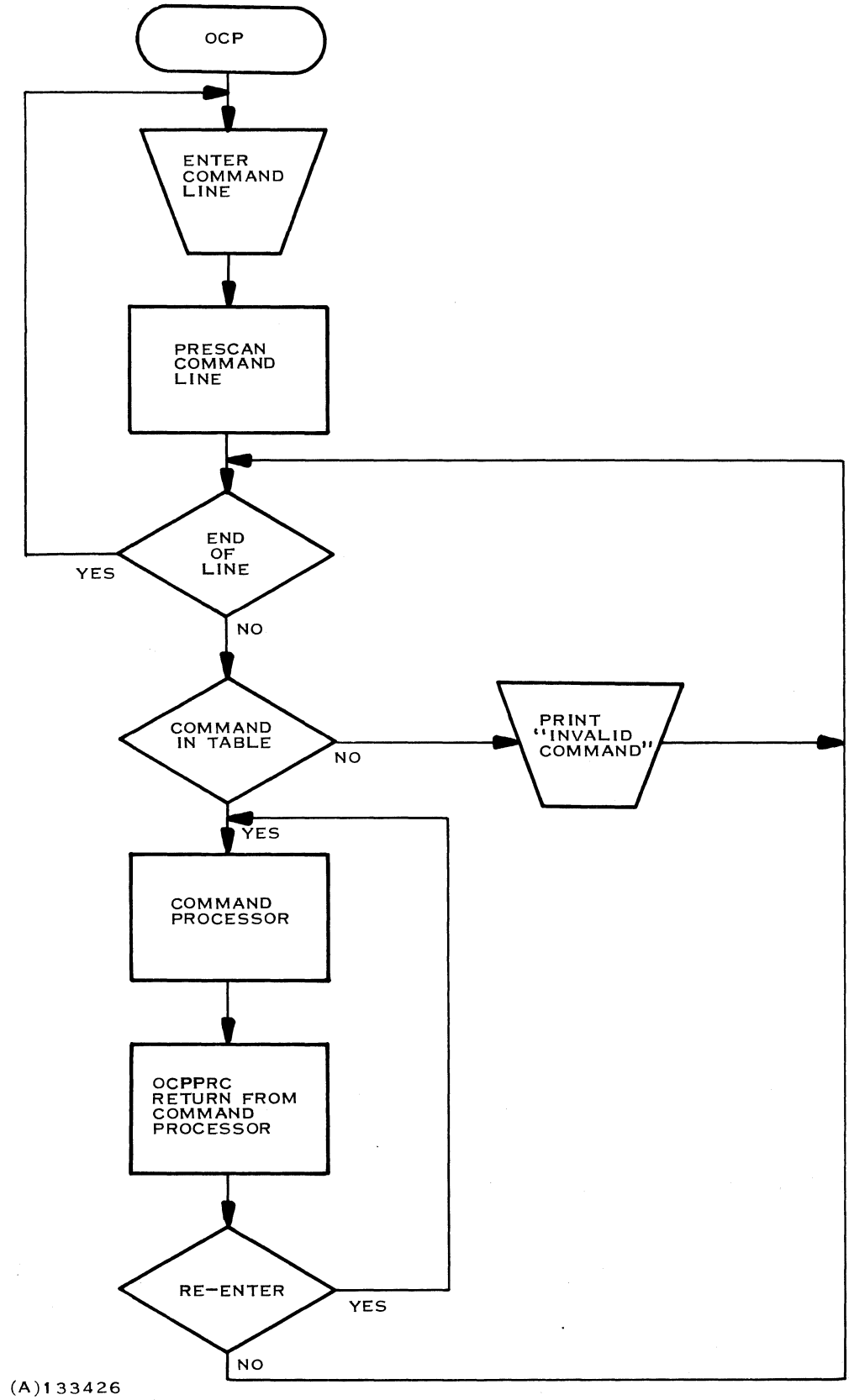


Figure 4-4. OCP Command Processing

If a command processor alters the contents of register 6, it must restore the contents before exiting. The command processor may use register 10 to obtain the operand of the command (or for any other purpose), but must place the address of any character within the command in register 10 before exiting. That is, register 10 may contain the address of the first character of the command word or the period that terminates the command, or any character between these characters. The command processor may use the other registers with no restrictions.

OCPPRC contains two subroutines that a command processor may call to obtain an operand. Subroutine GETARG is useful for obtaining decimal operands or those that contain characters, and subroutine GETHEX may be used to obtain hexadecimal operands.

Subroutine GETARG obtains the operand that starts in the byte following the byte addressed by register 10 and ends at a separator. GETARG places the operand right-justified in a six-byte string at location ARGBF2. The string is blank filled to the left. The word preceding the six-byte string may be used to build a supervisor call block for converting a decimal operand to binary. The location of the word preceding the string is ARGBUF. The calling sequence for subroutine GETARG is as follows:

```
BL      @GETARG
JMP     PERIOD
JMP     ERROR
 .
 .
 .
```

Branch to subroutine GETARG. Control returns to the word following the BL instruction when the operand contains only a period. When the operand contains more than six characters, control returns to the second word following the BL instruction. When the GETARG operation is successful, control returns to the third word following the BL instruction.

Upon return from subroutine GETARG, register 10 contains the address of the separator that follows the operand in the six-byte string at location ARGBF2.

Subroutine GETHEX obtains the hexadecimal operand that starts in the byte following the byte addressed by register 10 and ends at a separator. GETHEX converts the characters to a binary number and places the result in workspace register R0. The calling sequence for subroutine GETHEX is as follows:

```
BL      @GETHEX
JMP     PERIOD
JMP     ERROR
 .
 .
 .
```

Branch to subroutine GETHEX. Control returns to the word following the BL instruction when the operand contains only a period. When the operand contains an invalid hexadecimal digit, control returns to the second word following the BL instruction. Otherwise, control returns to the third word following the BL instruction.

If the OCP command processor has not altered register 15, a BL *15 instruction may be used to call GETHEX.

Upon return from subroutine GETHEX, register 10 contains the address of the separator that follows the operand returned in register 0.

Return from a command processor is to a return point in OCPPRC. One return point terminates the command with no output, three return points terminate the command with printed messages and another return point provides a specified I/O operation to a specified LUNO.

A return to location SCANPD passes control to OCPPRC for processing the next command with no output. This return is appropriate for commands that require no messages or results to be printed.

A return to location OCPRTN causes OCPPRC to print a message from buffer OUTBUF on LUNO 0 or LUNO 1 with optional additional processing. The option is specified by the terminating character placed in the buffer following the last character to be printed. The characters and the LUNOs and options are as follows:

- FC_{16} – print on LUNO 0, omit processing of any additional command on current command line.
- FD_{16} – print on LUNO 0, process next command.
- FE_{16} – print on LUNO 1, reenter command processor.
- FF_{16} – print on LUNO 1, process next command.

The first option, using terminator FC_{16}, is appropriate for an error condition because the remaining commands of the command line, if any, are skipped. The third option, using terminator FE_{16}, is appropriate for a repetitive command, such as a dump command. The other options return control to the next command.

When the FE_{16} terminator is used and the command processor is reentered following the printing of the message, the user must place an address in workspace register 3 before returning to location OCPRTN. Workspace register 3 must contain the address in the command processor at which it is reentered. OCPPRC alters the contents of registers 0, 1, and 2. The restrictions on the use of registers 6 and 10, described previously, apply. Otherwise the command processor may use the workspace registers as required.

A return to location ERRMSG causes OCPPRC to print an error message on LUNO 0. The error message is specified by an error code in workspace register 0. Control passes to the first command on the next command line, omitting any additional commands on the current line. The text of the error message may be in the command processor or in module OCPTBL. Table ERRTBL in module OCPTBL lists the error codes and the addresses of the associated error message texts.

A return to location ARGERR causes OCPPRC to print an error message on LUNO 0. The error message is as follows:

OPERAND ERROR(S)

Control passes to the first command on the next command line, omitting any additional commands on the current line.

A return to location IOURTN causes OCP to perform an I/O operation on a specified device and to process the next command. Workspace registers 8 and 13 define the operation and specify the device. The I/O operation code is placed in the most significant byte of register 8, and the LUNO is placed in the least significant byte. For forward or backward space operations, the number of records is placed in register 13. For other operations, the contents of register 13 are not significant. Since the interface with subroutine IOURTN has no provision for specifying a buffer, I/O operations that require a buffer (read or write, for example) are not valid. IOURTN performs an Open operation, the specified operation, and a Close operation, then processes the next command.

4.5.1 MODIFYING AN OCP COMMAND. To modify an OCP command, the user should obtain the source listing of the OCP modules from Texas Instruments, and modify the command processor to perform the operation in the desired manner. The subroutines and return locations in OCPPRC described in the preceding paragraph may be used as required.

4.5.2 ADDING AN OCP COMMAND TO A MODULE. The command processor for an OCP command must be a serially reentrant routine. As described previously, fourteen workspace registers of the workspace are available for temporary storage of data. When more space is needed, it must be provided in module OCPTSK by adding directives to provide the space at the end of the module. When the command processor detects an error for which one of the existing error messages is appropriate, the user may call subroutine ERRMSG with the error code for that message in R0. When the error requires a message unique to the command, the user should include the text of the message in the command processor, and list the error code and the address of the text in table ERRTBL in module OCPTBL. When an error requires an additional message common to several command processors, the text of the message should be placed in OCPTBL, and the error code and text address placed in table ERRTBL. Any exit from the command processor must be to one of the return points in OCPPRC.

A two-word entry for the new command must be added to the Command Word Table of the command processor module. The zero that terminates the Command Word Table must be placed following the additional entry.

4.5.3 ADDING AN OCP COMMAND MODULE. An OCP command module may contain one or more additional command processors, each of which must meet the requirements outlined in the preceding paragraph. The module must contain a Command Word Table consisting of a two-word entry for each command processor, terminated with a word that contains zero. The address of the Command Word Table in the added module must be placed in the Command Word Address Table in OCPTBL. The symbolic address of the Command Word Table in the added module must be added to the dummy word structure in OCPEND.

SECTION V

DATA STRUCTURES

5.1 GENERAL

This section describes the internal data structures of the TX990 operating system, including internal tables and file structures.

5.2 TXDATA

TXDATA contains most of the system data structures that change according to the specific system configuration (TASKDF contains the other data structures that vary.) TXDATA is generated in source statement form by the system generation program. The module must be assembled before being linked with the system. The first two words of TXDATA contain a branch instruction to the restart routine of the operating system. TXDATA should be the first module in the link to provide for ease in restarting the system.

5.2.1 DEVICE NAME TABLE. The device name table (DNT), generated at system generation time, is used by the system to map device names to devices. Each device included in the system has an entry in the DNT. Each entry, as shown in figure 5-1, consists of a pointer to the device PDT and a four character device name. The name is left justified in the field with trailing spaces. The dummy device has a zero value for the PDT address. The position of the entry in the table determines the internal unit number of the device.

5.2.2 LOGICAL DEVICE TABLE. Each LUNO assigned has a logical device table (LDT) associated with it. The LDT provides the link between the LUNO and the physical device or disc file. The LDT is built either by the system generation program when a LUNO is specified or by the file utility routine (FUR) when an assign LUNO supervisor call is made. When a LUNO is assigned, a buffer is allocated from the buffer pool for this LDT. There are two kinds of LDTs: device and file. A device LDT, shown in figure 5-2, maps a LUNO to a device; and a file LDT, shown in figure 5-3, maps a LUNO to a particular file on a disc.

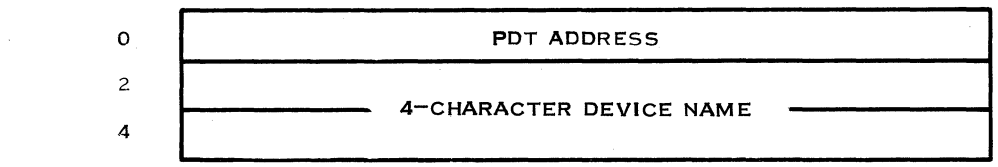


Figure 5-1. Device Name Table (DNT) Entry

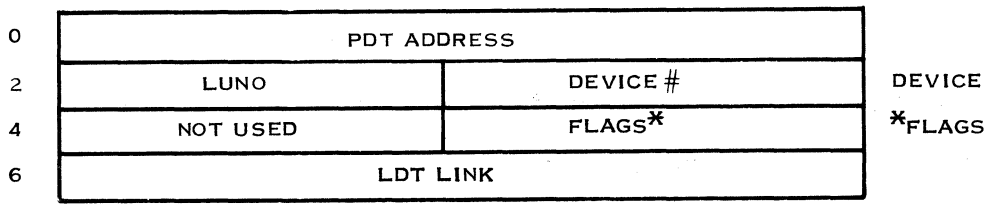


Figure 5-2. Device LDT

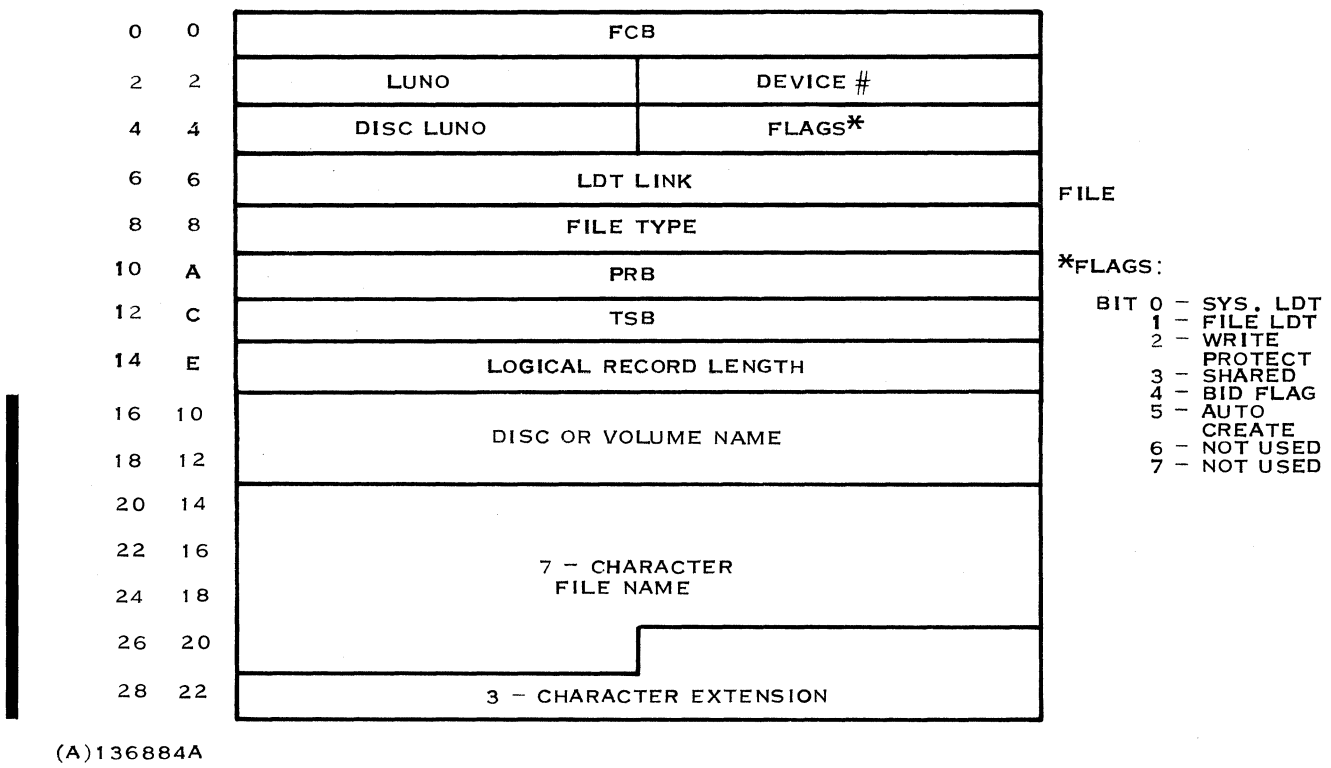


Figure 5-3. File LDT

5.2.2.1 **Device LDT.** Bytes 0-1 contain the address of the PDT for the device to which the LUNO is assigned.

Byte 2 contains the actual LUNO.

Byte 3 contains the device number (positional entry in the DNT).

Byte 5 contains flags. Bit 0, when set, indicates that the LUNO is a system LUNO and may not be released or reassigned. Bit 1 will be a zero to indicate a device LDT. All other bits are ignored.

Bytes 6-7 contain a link to the next LDT in the chain.

5.2.2.2 **File LDT.** When the file is opened, bytes 0-1 contain the FCB address of the file in memory. When the file is not open, this field is zero. When the file is in the process of being opened, this field is a negative one.

Byte 2 contains the actual logical unit number (LUNO).

Byte 3 contains the disc device number (positional entry in the DNT).

Byte 4 contains the LUNO assigned to the disc drive on which the file is resident.

Byte 5 contains flags. Bit 0, when set, indicates that the LUNO is a system LUNO and cannot be released or reassigned. Bit 1 will be set to one to indicate that this is a file LDT. Bit 2 is set to one by file management in the open processor when the file is write protected. Bit 3 is set to one by file management in the open processor when the file may be shared with other tasks. Bit 4 is the rebid flag and, when set, indicates that the task should be rebid when file management finishes processing the current I/O call. Bit 5 is set by FUR in the assign LUNO operation to indicate that the file should be created at open time if it does not exist. Bit 6 indicates that volume names are being used.

Byte 6-7 contain the link to the next LDT in the chain.

Bytes 8-9 contain the file type. A zero indicates a sequential file and a two indicates a relative record file.

Bytes 10-11 contain the SCT address +2 of the current I/O call. This field is zero when no calls are pending or being processed for this LUNO.

Bytes 12-13 contain the TSB address of the task issuing the I/O call.

Bytes 14-15 contain the logical record length of the relative record file. This field is used only when a relative record file is auto-created at open time. This field is zero for a sequential file.

Bytes 16-19 contain the diskette or volume name.

Bytes 20-26 contain the seven character file name. The name is left justified in the field and blank filled.

Bytes 27-29 contain the three character extension name. The name is left justified in the field and blank filled.

5.2.3 BUFFER POOL. TX990 maintains a pool of internal buffers which are dynamically allocated and deallocated by buffer management for use as logical device tables, file blocking buffers, intertask communications messages, etc. The size and number of these buffers are specified at system generation time. A separate list is maintained for buffers of each size. A header table contains an entry for each buffer size that points to the beginning of the respective list. Figure 5-4 shows the structure of the header table and figure 5-5 shows the linkage between buffers.

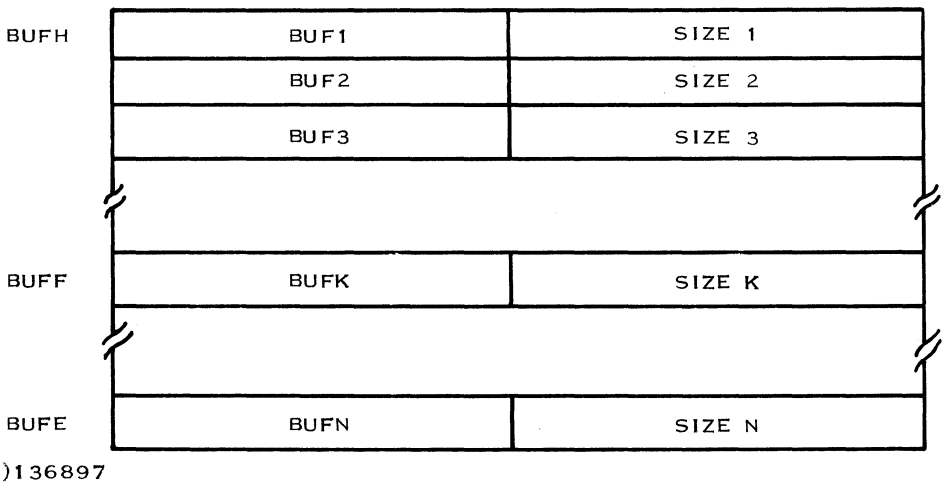


Figure 5-4. Buffer Header Table Structure

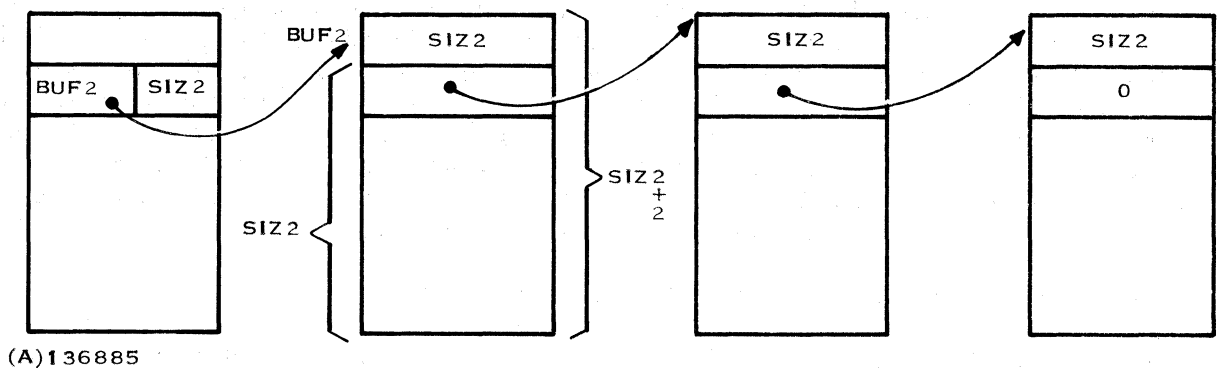


Figure 5-5. Buffer Linkage

5.2.3.1 **Header Table.** For each size of buffer, there is a two word entry in the header table. Word 1 points to the buffer at the top of the list and the second word is the buffer size. BUFH is the label of the first entry and BUFE is the label on the last entry in the table. BUFF is the label of the entry associated with the buffer created during SYSGEN as a "default buffer". A call to buffer management for a fixed length buffer allocates a buffer from this list. The communication package uses this buffer size.

5.2.3.2 **Buffer Linkage.** Preceding each buffer is a data word containing the size of the buffer. This word is used to determine which list the buffer belongs on when the buffer is deallocated. The first word of the buffer is the link to the next buffer and points to the word preceding that buffer. The last buffer in the chain has a zero link.

5.2.4 **FMPBUF.** When a disc is included in the system at system generation time, the system generation program generates a buffer called FMPBUF. This buffer is used by file management and the file utility routine as a temporary blocking buffer for use with create, compress, delete, etc. The use of FMPBUF is restricted by the flag FMPFLG. FMPBUF is 224 bytes in length for the diskette system.

5.2.5 **PHYSICAL DEVICE TABLES.** Each device in the system that can be accessed by a LUNO has a physical device table (PDT) associated with it. The PDT contains information used by the operating system to support the device. All PDTs are chained together with the address of the first PDT in the chain contained in the word PDTSTR. The system table contains this pointer. No words initialized by the system generation program can be modified by the DSR unless otherwise stated. The size and contents of the PDT vary according to the particular device.

The PDT consists of three parts: device service routine (DSR) workspace, device information block, and DSR temporary storage. The DSR workspace and device information block are of fixed size; the DSR temporary storage area is variable and dependent on the device.

5.2.5.1 **DSR Workspace.** The first 16 words of the PDT (see figure 4-1) are used as the workspace for the DSR when an I/O call is used. Some of the registers are used as scratch registers by the DSR and some contain device-related information used by both the DSR and the operating system. The registers are as follows:

- Register 0 – Used by the system to link PDTs. Register 0 may not be modified by the DSR.

- Register 1 – Contains the address of the second word of the supervisor call block (SCB). The operating system places this value in register 1 prior to entering the DSR.

- Register 2 – Device status word. Contains information about the current status of the device. Bit 2, the initiate I/O flag, is set by the operating system in response to an initiate I/O call and cleared when the operation is completed. Bit 4 is the close device flag. If set by the operating system, it causes the device to be deassigned when the operation terminates. This bit is always set for a record-oriented device. For file-oriented devices, this bit is set for close and abort operations only. Bit 5, the reentry flag, is set by the DSR to cause the DSR to be reentered at its interrupt entry point when the current system time interval expires. Bit 7 is used by communication devices to differentiate between abort LUNO and abort SCB supervisor calls. Bits 12-15 contain the interrupt level, minus one, of the device.

- Register 3 – The device characteristics word. Contains static information about the device. This word is generated at system generation time and should not be altered during execution. Bit 0, when set, specifies that the device is file oriented. Bit 1, when set, designates the device as a TILINE device rather than CRU device. When bit 2 is set, the system will time-out I/O operations to a device. Bit 3, when set, specifies that a task must be privileged to access the device. Bit 4 designates this device as the system console. Bit 5 specifies that this device is a communication device and that special handling is required. Bit 6, when set, indicates that the device is a 911 VDT rather than a 913 VDT. Bits 12-15 contain the device type code.

- Register 4 – contains the address of the device information block. This address is generated by the system generation program and should not be modified.

- Register 5 – Contains the keyboard status block (KSB) address when the device is a VDT or contains the address of the multiunit workspace for a device with multiple units per controller. When neither of these conditions apply, this is a scratch register for DSR usage only. The KSB or multiunit workspace address is supplied by the system generation program.

- Register 6 – Normally contains the internal DSR reentry branch address used by the interrupt routine to branch within the DSR. It may also be used as a scratch register.

- Registers 7-11 – DSR scratch registers.

- Register 12 – Contains the CRU base address or TILINE address of the device supplied by the system generation program.

- Registers 13-15 – Contain the return vectors.

5.2.5.2 **Device Information Block.** The device information block (figure 5-6) contains information supplied and used by the operating system and DSR to control the device I/O. The block is a fixed length for all PDTs, with the following entries:

Bytes 0-1 – Contain the TSB address of the task to which the device has been allocated. The device is not allocated when this field is zero.

Bytes 2-3 – Contain the address, supplied by the system generation program, of the DSR.

Bytes 4-5 – Contain the timeout count specified at system generation time.

Bytes 6-7 – Contain the number of system-time intervals remaining before the device will timeout. This field is initialized to the timeout count in bytes 4-5 before the DSR is called by the operating system.

Bytes 8-9 – Contain the interrupt entry point into the DSR. This field is supplied by the system generation program.

Bytes 10-13 – Contain the first and last entry pointers of the PDT queue. The entries in the PDT queue are the TSBs of the tasks requesting the services of the DSR. This queue is used by the operating system to control access to the DSR.

Bytes 14-15 – Contain the address of the LDT currently being processed by the DSR. This field is zero when the DSR has finished processing all I/O calls.

Bytes 16-17 – The busy flag. Set to FFFF when the DSR is processing an I/O call.

Bytes 18-21 – Contain the workspace and status of the operating system prior to the "BLWP" to the DSR. These values are saved to ensure no loss of content if an interrupt occurs while the DSR has interrupts enabled.

5.2.5.3 **DSR Temporary Storage.** This block contains information that is used by the DSR only. The size of the block varies according to the device. The system generation program has an internal table used to generate the block along with the initial values. The user must supply this block for nonstandard devices. Some devices (i.e., line printer and card reader) do not require any temporary storage.

5.2.6 **MULTIUNIT WORKSPACE.** Some DSRs support more than one device (i.e., disc per controller). These DSRs require, in addition to one PDT per device, a single workspace to identify the device causing the interrupt (see figure 5-7). The multiunit workspace is created by the system generation program for any standard devices requiring this workspace. When an interrupt occurs, the DSR interrupt decoder is entered and this workspace is used.

Registers 0– (N-1), where N is the number of devices per controller, are the PDT addresses of each device supported by the controller. (For a diskette controller, there are a maximum of four PDT addresses.)

Register 12 is the CRU or TILINE address of the device. This address is supplied by the system generation program.

Registers 13-15 contain the workspace, PC, and status of the program interrupted.

All other registers are scratch registers for decoder usage.

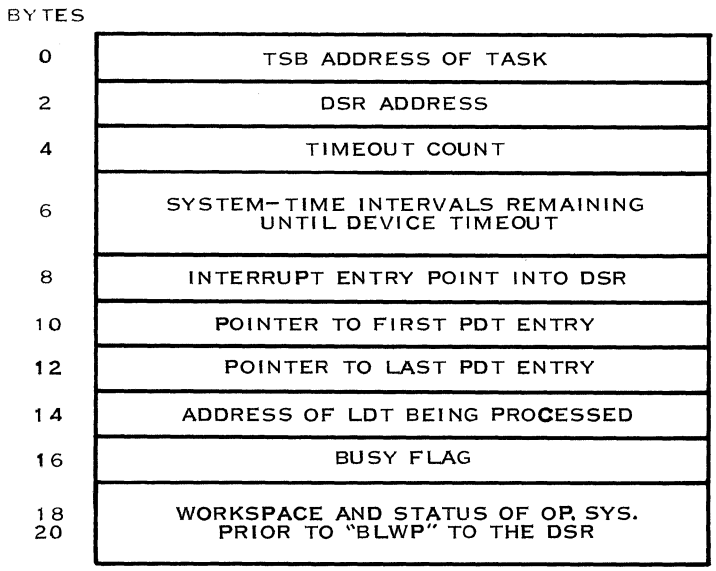


Figure 5-6. Device Information Block

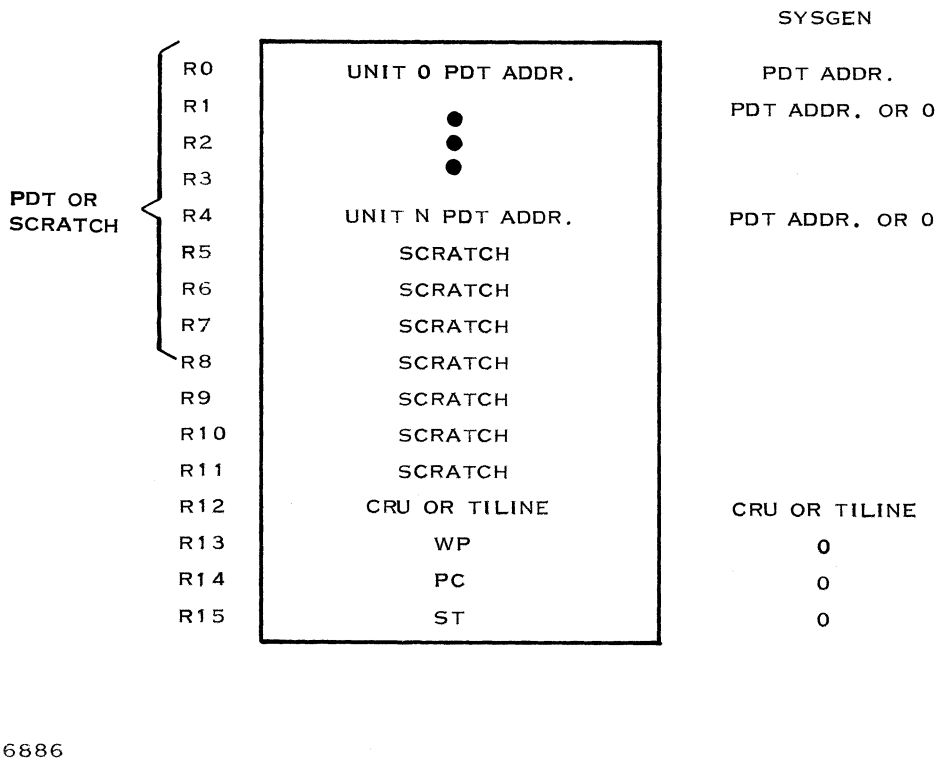


Figure 5-7. Multiunit Workspace

5.2.7 **KEYBOARD STATUS BLOCK.** Each VDT in the system requires a keyboard status block (KSB) for the workspace of the keyboard interrupt as shown in figure 5-8. The KSB, in addition to being a workspace for the interrupt handler, contains a queue of the characters input from the keyboard. If the mode flag, register 0, contains a nonzero value, the VDT is not allocated to a task. The mode flag is the TSB address of the task if the VDT is allocated.

- Register 1 points to the beginning of the character queue.
- Register 2, the input pointer, points to the location within the character queue at which to store the next character entered.
- Register 3, the output pointer, points to the next character to be retrieved by either the DSR, Get Character Supervisor call, or VDT utility.
- Registers 4-6 are the character queue.
- Register 7, byte 0, is used by the 911 VDT as a constant, $C0_{16}$, and marks the end of the character queue. For the 913 VDT, this byte is the last character in the queue.

  Register 7, byte 1, contains flags used by the interrupt handler and DSR. Bit 0 must be a 1 to indicate the end of the character queue, although it is not needed for the 911 VDT. Bit 1, when set, indicates that the device may be halted (by entering the space key) or aborted (by entering the escape or reset key). This bit is set by the DSR when displaying data. Bit 2, when set, directs the DSR to halt the display of any new characters. Bit 3, when set, directs the DSR to abort the current display operation. Bits 4, 5, and 6 are not used, and are set to zeros. Bit 7 is set by the system generation program to indicate a 911 VDT.
- Registers 8-11 are scratch registers for station handler usage only.
- Register 12 contains the VDT CRU address + 10_{16}. This field is generated during system generation.
- Registers 13-15 contain the content of the interrupted task (i.e., WP, PC, and ST).

A list of the KSBs in the system in maintained in the KSB table. The table consists of the KSB addresses listed sequentially according to station number. The table is terminated with a zero entry.

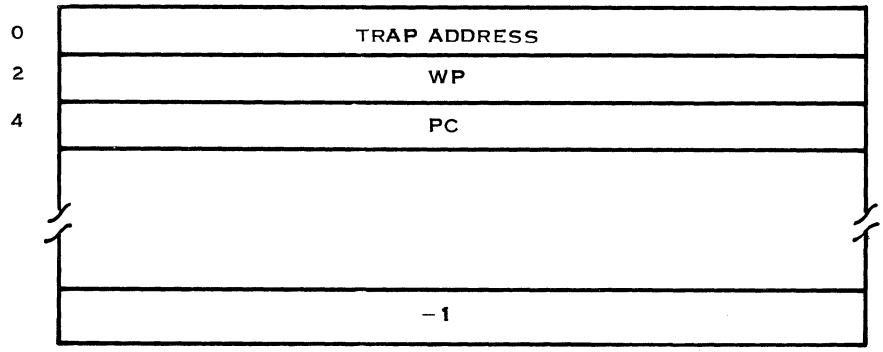
5.2.8 **INTERRUPT VECTOR TABLE.** For each interrupt defined at system generation time, an interrupt vector is created. Each entry in the interrupt vector table, shown in figure 5-9, contains the hardware trap vector address and the PC and WP to be placed at that address. The hardware trap vector address is the actual memory address from which the hardware interrupt logic retrieves the interrupt branch vectors. The PC points to the entry point in the interrupt decoder for this interrupt (described below). These vectors are placed in the hardware interrupt trap locations when the system is initially loaded. The interrupt vector table does not contain the interrupt vectors of the system interrupts, 1 (powerup), 2 (CPU), 5 (clock). These vectors are contained in TXEND. The interrupt table is terminated by a negative one in the trap address field.

BYTE DEC.	NUMBER HEX	USE	SYSGEN INITIALIZATION
0	0	MODE FLAG	0
2	2	QUEUE ADDRESS	DATA $+6
4	4	INPUT POINTER	DATA $+4
6	6	OUTPUT POINTER	DATA $+2
8	8	CHARACTER QUEUE	0
10	A	CHARACTER QUEUE	0
12	C	CHARACTER QUEUE	0
14	E	CHAR QUEUE / FLAGS	>80 (913) >C081 (911)
16	10	SCRATCH REGISTER - R8	0
18	12	SCRATCH REGISTER - R9	0
20	14	SCRATCH REGISTER - R10	0
22	16	SCRATCH REGISTER - R11	0
24	18	KEYBOARD CRU BASE	CRU BASE
26	1A	SAVED WP	0
28	1C	SAVED PC	0
30	1E	SAVED ST	0

FLAGS
8 - MUST BE A 1 FOR VDT 913
9 - HALT/ABORT ACTIVATED
10 - HALT DISPLAY
11 - ABORT DISPLAY
15 - VDT 911 KSB

(A)136887

Figure 5-8. TX990 KSB



(A)136888

Figure 5-9. Interrupt Vector Table

5.2.9 INTERRUPT DECODER. For each interrupt defined at system generation time, an interrupt decoder is created. The interrupt decoder determines which device caused the interrupt when multiple devices exist on the same interrupt level. The decoder contains the workspace address and entry point for each of the individual device interrupt routines. After the DSR returns to the interrupt decoder, it branches to a common return in the task scheduler (TXROOT). Figure 5-10 is an example of an interrupt decoder with interrupts 3 and 6 defined. Interrupt 6 has two devices on the interrupt level.

5.2.10 USER-DEFINED SVC TABLE. When a user-defined supervisor call is defined at system generation time, an entry is created for it in the user SVC table. The user SVC table consists of a two-byte entry for each supervisor call. Each two-byte entry is the entry address into the processor for that supervisor call. The supervisor call code of an entry is relative to the beginning of the table. The first entry corresponds to SVC 80_{16}. The second corresponds to SVC code 81_{16}. Space in the table is allocated starting with SVC code 80_{16} to be the largest SVC code defined. The entry for any SVC code within the table that has not been defined will be initialized to the address of the illegal SVC processor, ILLSVC.

5.2.11 INTERTASK MESSAGE QUEUE. The Put Data supervisor call is used to send messages and/or data to other tasks. The message is placed in the Intertask Message Queue and may be retrieved by the Get Data supervisor call. (The buffer for the message is obtained from the buffer pool. Figure 5-11 shows the format of the message on the queue.

Bytes 0-1	contain the queueing link.
Bytes 2-3	contain the size of the buffer allocated for the message.
Byte 4	is not used.
Byte 5	contains the message ID.
Bytes 6-7	contain the number of characters in the message.
Bytes 8-n	contain the message.

5.3 TASK DEFINITION

The system generation program creates a task definition module, TASKDF. TASKDF consists of task status blocks linked together. TASKDF is a source module and must be assembled before being linked with the system. The TSBs for all tasks linked with the system, and the TSB for the first dynamic task are contained in this module. In a multiple dynamic task system, TSBs for subsequent tasks are allocated from a buffer pool defined in the TXDATA module at system generation. Refer to paragraph 5.2.3 for a description of buffer pool allocation.

```
        PAGE
*       INT DECODER
        REF    TRAPPRT XWS, BADINT
WSP4    EQU    $-18
        REF    CRINT
        DATA   PDT6, CRINT, 0, 0, 0, 0, 0
WSP6    EQU    $-18
        DATA   0, 0, 0, 0, 0, 0, 0
LEV6    LI -   11, TAB6
        CLR    10
GET6    MOV    *11+, 12
        JEQ    XIT6
        TB     0
        JNE    NXT6
        BLWP   *11
        INC    10
NXT6    AI     11, 4
        JMP    GET6
XIT6    MOV    10, 10
        JEQ    BAD
        JMP    RET
TAB6    EQU    $
        REF    GO
        DATA   >1E, PDT2, GO
        REF    LPINT
        DATA   >7E, PDT5, LPINT
        DATA   0
WSP7    EQU    $-18
        REF    FPYDCD
        DATA   XWP0, FPYDCD, 0, 0, 0, 0, 0
XWP0    DATA   PDT0
        DATA   PDT1
        DATA   0
        DATA   0
        BSS    16
        DATA   >A0
        BSS    6
LEV0    BLWP   9
        JMP    RET
BAD     BLWP   @TRABAD
RET     B      @TRAPRT
TRABAD  DATA   XWS, BADINT
        PAGE
*       INT    VECTORS
        DEF    TRPINT
TRPINT  EQU    $
        DATA   >10, WSP4, LEV0
        DATA   >18, WSP6, LEV6
        DATA   >1C, WSP7, LEV0
        DATA   -1
```

Figure 5-10. Interrupt Decoder Example

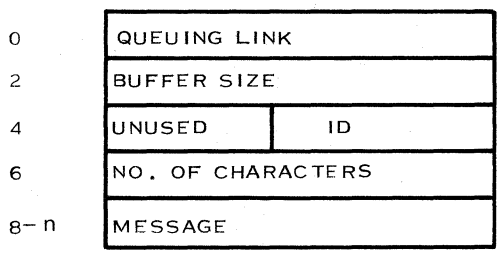


Figure 5-11. Intertask Message Format

5.3.1 **TASK STATUS BLOCK.** Each task linked with the system requires a task status block (TSB) as shown in figure 5-12. The TSB is used to control the execution of the task.

Bytes 0-1 – Contain the queuing link, which is used when the TSB is queued. A TSB may be queued on the active queue, file management queue, diagnostic queue, device queue, etc.

Bytes 2-7 – Contain the current WP, PC, and ST of the task. These bytes are updated whenever a task is preempted or suspended. The initial values of WP and PC are retrieved from the first two words of the task.

Byte 8 – Contains the priority level, 0-3, of the task. Priority is represented as follows:

0	– Priority 0
>81 to >FF	– Real-time Priorities 1 to 127
1-3	– Priority 1-3

Byte 9 – Contains the current state of the task. Possible task states are listed on figure 5-13.

Byte 10 – Contains task status flags. Bit 0, when set, indicates that the task is privileged. All tasks linked with the system are privileged. Bit 3 indicates that the task contains an end action vector in the third word of the task. When a task error occurs, the system branches to the user's end action routine. If this bit is not set, a fatal task error causes the diagnostic task to be called. Bit 6, when set, indicates that the task is being aborted, either from an end program or end task supervisor call or from a kill task OCP command. Bit 7, when set, indicates that an Unsuspend supervisor call was issued to a task that was not suspended. The next Unconditional Suspend supervisor call is overridden by this set bit and the bit cleared. Bits 2, 4, and 5 are not used and are set to zeros.

Byte 11 – Contains the task I.D.

Bytes 12-13 – Contain the beginning address of the task.

Bytes 14-15 – Used for temporary storage. When a task is active, the Task Sentry timer value is maintained here. When the task is in a time delay, this word contains the number of system-time intervals remaining in the delay. When a fatal task error occurs, the task error code is passed via this word to the diagnostic task. The LDT address is passed via this word when file management is called.

Bytes 16-17 – Contain the static link to the next TSB in the chain. This field will be zero in the last TSB in the chain.

Byte 18 – Contains the initial task state. Bits 0-3 contain the initial status of the task when the operating system is loaded, restarted, or at powerup. Bit 0, when set, indicates the task is privileged. Bit 1, when set, indicates that the task is a system task. Bit 2, when set, indicates the task is to be executed at power restart. Bit 3, when set, indicates the task is to be executed when the operating system is loaded or at restart. Bits 4 through 7 are reserved.

Byte 19 – Contains the ID of the attached procedure. This field is 0 if no procedure is attached.

Bytes 20-27 – Used for temporary storage. When a task is queued for I/O, the call is re-executed from the TSB. A two-word XOP instruction and a two-word branch to return to the task are stored here and then executed. When a task is bid, the task parameters are stored in bytes 24-27. A Get Parameters supervisor call retrieves the parameters from this area.

Bytes 28-31 – These bytes are present in the TSBs of dynamic tasks only when the multiple dynamic task option is specified during system generation. Bytes 28-29 contain a pointer to a linked list of dynamically allocated memory blocks for the task. Bytes 30-31 are reserved and must be zero.

5.4 TXROOT

The data structures contained in TXROOT are those that are configuration independent (i.e., structures that do not vary according to the devices or modules included in the system). These structures include queue headers, supervisor call table, and flags.

5.4.1 SYSTEM TABLE. The system table, shown in figure 5-13, consists of pointers to other structures in the system. This table may be accessed by privileged tasks through the get system table supervisor call. The system table consists of the following entries:

Bytes 0-1 – Contain a pointer to the time and date block. The time and date block consists of Julian year, day, hour, minute, and second in that order.

Bytes 2-3 – Point to the first TSB in the TSB chain.

Bytes 4-5 – Point to the first PDT in the PDT chain.

Bytes 6-7 – Point to the first LDT in the LDT chain.

Bytes 8-9 – Point to the default disc name. The default disc name is generated by the system generation program.

Bytes 10-11 – Point to the default printer name. The default printer name is generated by the system generation program.

Bytes 12-13 – Point to device name table. The device name table associates a logical name to type device.

Bytes 14-15 – Point to the first PSB in the PSB chain.

5.4.2 SYSTEM FLAGS. TXROOT contains a number of words used as flags by the system. These flags are used to force exclusive access to a resource (routine or buffer) and are set to negative one when the operating system is loaded and at system restart to indicate that the resource may be accessed. The user executes a test and set (ABS instruction) on the flag to gain access to the resource. The flag should be reset by the user to negative one when the resource is released. The three flags are FMPFLG, FURFLG, and LDRFLG. FMPFLG controls access to FMPBUF. FURFLG controls access to routines in FUR. LDRFLG controls access to the task loader routine.

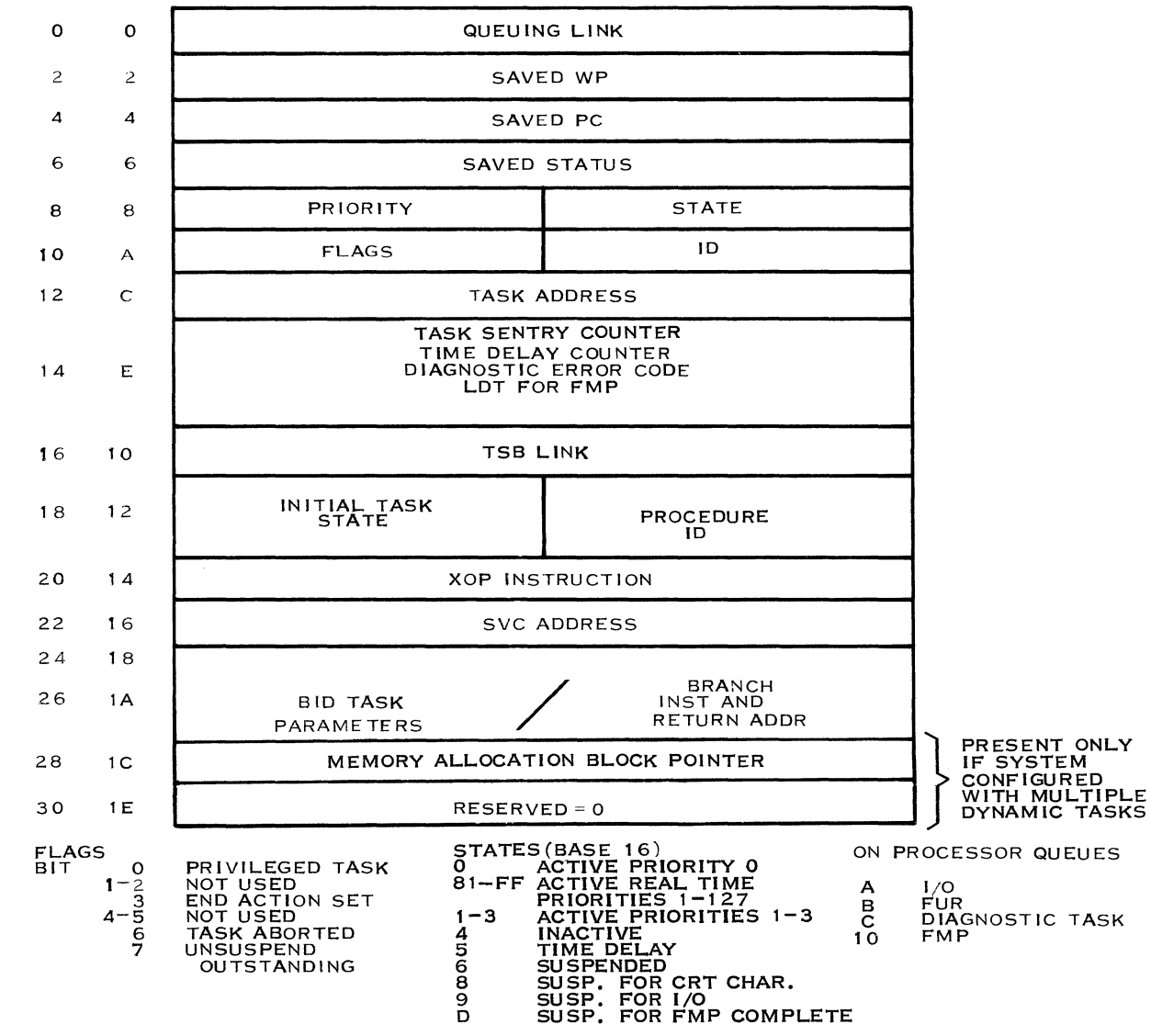


Figure 5-12. Task Status Block

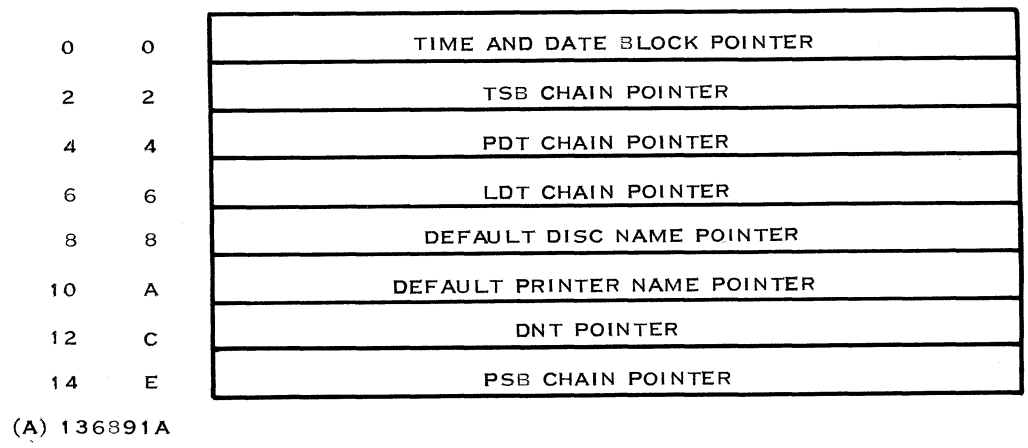


Figure 5-13. System Table

5.4.3 QUEUES. The basic structure used by the operating system to control the execution of tasks is the queue. The queue is used to force a first-in/first-out (FIFO) order to tasks requiring the services of a particular system task, DSR, and to force a priority ranking of tasks requesting the task scheduler. The queueing is handled by two routines in TXROOT. A task is queued by placing its TSB on a queue. The queue consists of a queue header and the linked TSBs.

5.4.3.1 Active Queue. The active queue consists of two linked lists of active tasks in order of their priorities. Tasks with priority 0 and real-time priorities 1-127 are on the first list, and tasks with priorities 1-3 are on the second list. The highest priority task in each group is found at the top of the respective list. The scheduler always places the highest priority active task into execution when the scheduler is executed. When a task becomes active due to a bid task, upon I/O completion, etc., the appropriate list is searched from top to bottom, if needed, to determine where to place the task on the active queue. The task is placed above lower priority tasks on the queue and below tasks of equal or higher priority. Therefore, tasks of equal priority are treated in a first-in/first-out (FIFO) manner.

5.4.3.2 System Task Queues. Tasks that require the services of file utility, file management, or the diagnostic task are suspended and placed on the required system task's input queue. Once bid, the system task continues to service requests on its queue until it is empty, at which time the system task terminates.

5.4.3.3 DSR Queue. When a DSR is busy and another I/O call is made, the task issuing the call is placed on the device's queue. The queue headers are located in the device information block of the device PDT.

5.4.3.4 Intertask Message Queue. When a task issues a Put Data supervisor call, the message is placed at the end of the Intertask Message Queue. When a Get Data supervisor call is issued, the queue is searched linearly for the oldest message with the specified message identification number.

5.4.4 SUPERVISOR CALL (SVC) TABLE. The supervisor call table provides the entry points to the individual supervisor call processors. The SVC table consists of two parts: a list of the SVC code range for each block of consecutive SVC operation codes and the actual blocks of entry points.

5.4.4.1 Supervisor Call Table List. The supervisor call table list, shown in figure 5-14, contains a four-byte entry for each noncontiguous block of SVC codes. Bytes 0-1 contain the upper and lower SVC code of the block, respectively. Bytes 2-3 contain the starting address of the supervisor call table block. The SVC table list is terminated with a zero in bytes 0-1.

5.4.4.2 Supervisor Call Table Block. The supervisor call table block, shown in figure 5-15, consists of a number of words which contain the address of each SVC processor entry point. Each word in the block corresponds to an SVC code within the range defined by the SVC table list.

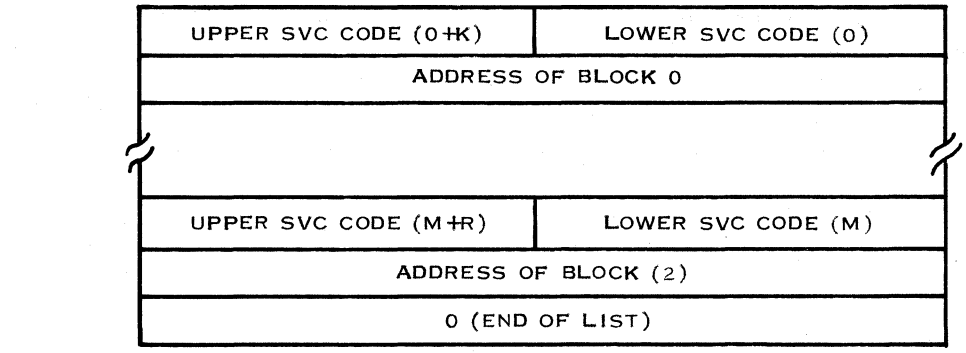


Figure 5-14. Supervisor Call Table List

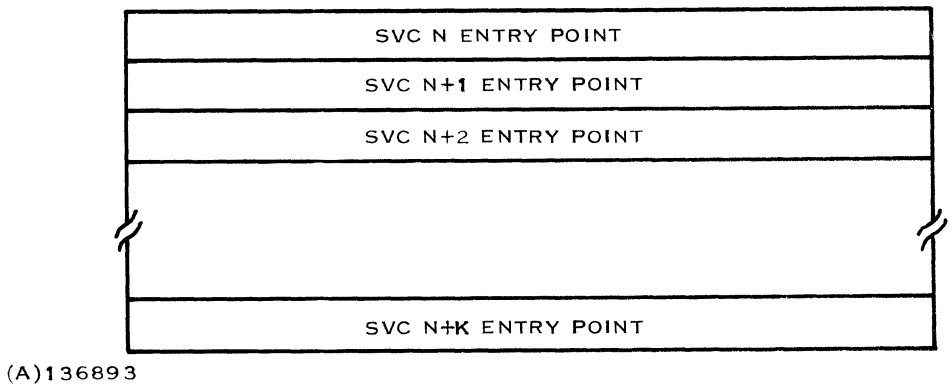


Figure 5-15. Supervisor Call Table Block

5.5 PHYSICAL DISKETTE STRUCTURE

The diskette consists of a number of physical tracks (.77 for a one-sided, single-density IBM-compatible diskette) with 26 sectors per track. The diskette is divided into logical units called allocation units. Each allocation unit consists of 6 sectors and there are 333 allocation units per diskette. The physical record length is the block of data read from or written to the diskette at one time. The diskette is formatted with one physical record per sector.

5.6 LOGICAL DISKETTE STRUCTURE

The first six allocation units are reserved for system usage and the others are available for user files. The diskette has the following structure:

Allocation Unit	Sectors	Use
0-3	All	Boot loader
4	0	System information block
4	1	Allocation table
4	2	Bad allocation table
5	0-5	Directory
6-333	All	File allocation

5.6.1 BOOT LOADER. Allocation units 0-3 contain the system boot loader (TXBOOT program). This program is loaded from the diskette/cassette ROM loader; in turn, it loads the system file. The boot loader is not required for a diskette which is to be used as a data diskette only; i.e., will not contain a system file. The TXBOOT program is stored in memory image format. The last word of each sector is a flag to the ROM loader. When the last word is not equal to negative one, the ROM loader loads the next sector. When it is equal to negative one, the ROM loader terminates and executes the TXBOOT program. TXBOOT is copied to the diskette in consecutive sectors beginning at physical track 0, sector 0. All of the sectors used are contained in allocation units 0-3.

5.6.2 DISC INFORMATION BLOCK. The disc information block, shown in figure 5-16, contains configuration data needed by the system and utilities. The format of the disc information block is:

Bytes 0-31 – Disc identification field. This information is supplied by the user during disc initialization (BACKUP). Typically, this field contains a diskette identification name and date. This field may be changed using the disk load (DL) command of SYSUTL or DSKDMP.

Bytes 32-33 – System file address. This field contains the starting allocation unit of the file designated as the system file (designated by use of the SF command of SYSUTL). The field contians zeros if no system file has been designated.

Bytes 34-35 – Contain an ASCII "TX" and is used to indicate that the disc was initialized. The backup utility examines these bytes on the output diskette before using the diskette.

Bytes 36-37 – Reserved.

Bytes 38-41 – Contain the four character volume name. When no volume name is used, it is binary 0. This field is initialized during disc initialization (BACKUP) and may be modified using the change volume name (CV) command of SYSUTL.

Bytes 42-43 – Reserved for future expansion.

5.6.3 ALLOCATION BIT MAP. The allocation bit map (allocation unit 4, sector 1) is a bit map used to indicate the availability of an allocation unit. A 1 in a bit position indicates that the allocation unit associated with that bit is allocated. The first bit in the map (bit 0) is associated with allocation unit 0, the second bit (bit 1) is associated with allocation unit 1, etc. The allocation bit map is initialized during disc initialization to all zeros. The bits corresponding to the system-reserved allocation units, 0-5, unused bits in the sector (those greater than bit position 332), and bits associated with bad allocation units are set to 1.

5.6.4 BAD ALLOCATION BIT MAP. The bad allocation bit map (allocation unit 4, sector 2) is used to mark defective allocation units on the diskette. Utility programs use this bit map to determine bad units. This table is identical in size and format to the allocation table. During diskette initialization, when a defective allocation unit is found, the appropriate bit is set to 1 in both the bad allocation bit map and allocation bit map. The bit in the allocation bit map is set so that the allocation unit will not be used. The unused bits in the sector (those greater than bit position 332) are set to 1s.

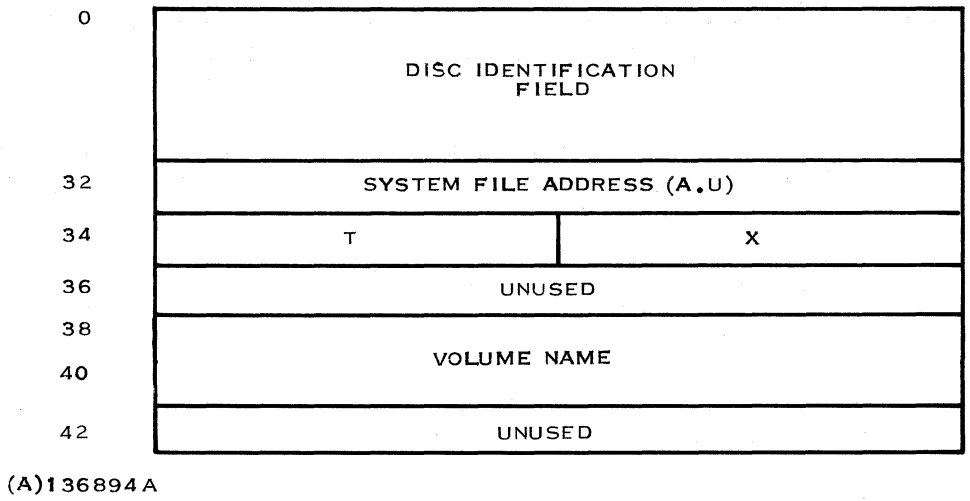


Figure 5-16. Disc Information Block

5.6.5 DIRECTORY. The directory (allocation unit 5, sectors 0-5) is used by the system to map file names to the physical start location of the files on the diskette. Each of the six sectors of the directory has space for eight files, allowing a maximum of 48 files on the diskette. Each file entry, as shown in figure 5-17, is 16 bytes long and has the following format:

Bytes 0-6 – Contain the 1-7 character file name. The name is left justified in the field and blank filled.

Bytes 7-9 – Contain the 1-3 character extension name. The extension is left justified and blank filled. This field is all blanks when the file does not have an extension. The combination of file and extension name uniquely define a file on the diskette.

Bytes 10-11 – Contain the number of the starting allocation unit.

Byte 12 – The protection code is an ASCII character U, D, or W, which defines the file as unprotected, delete protected, or write protected. This field is modified by the change protection (CP) operation of SYSUTL or under program control.

Byte 13 – File type, is a hexadecimal number that represents the type of file. A sequential file has a value of 0 and a relative record file has a value of 2.

Bytes 14-15 – Are reserved for further expansion.

When the diskette is initialized, the directory is initialized to periods ($2E_{16}$). When a file is created the file entry is placed in the first available space in the directory. When a file is deleted, the entry is overwritten with periods.

5.7 LOGICAL FILE STRUCTURE

For each file on the diskette there exists a file control block (FCB) which contains a list of allocation units allocated to the file, and other data. All files begin at physical record 0 of an allocation unit. The first physical record (0) contains the FCB. The actual file data begins at physical record 1. TX990 supports two types of files, sequential and relative record. Each file type has a different record and file format.

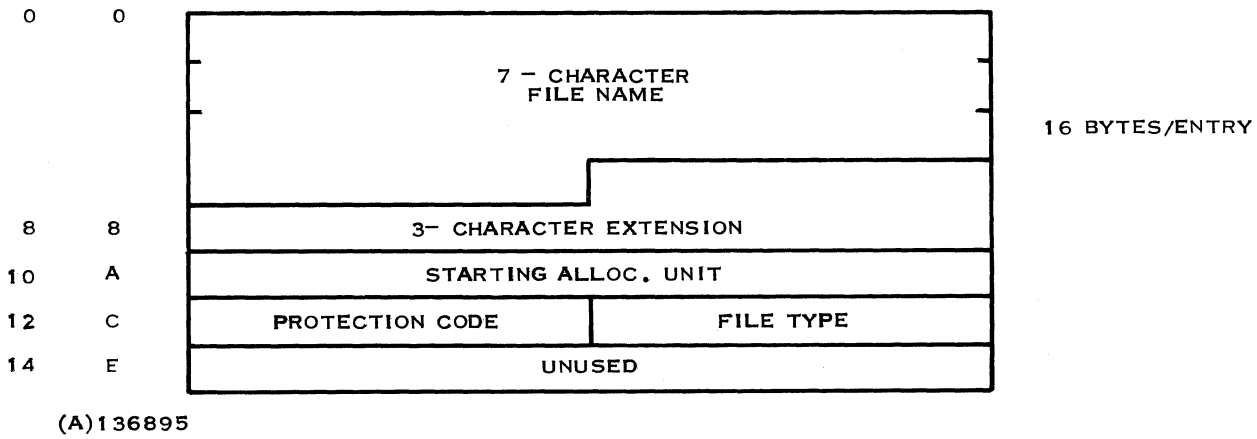


Figure 5-17. Directory Entry

5.7.1 SEQUENTIAL FILE

5.7.1.1 Record Format. A sequential logical record consists of N characters followed by an end-of-logical-record character. There are also control characters for end-of-physical-record, end-of-file, and blank compression. The characters are defined as follows:

FF	End-of-logical-record
FE	Blank compression (to be followed by a one-byte count)
FC	End-of-physical record
FB	End-of-file

To avoid conflicts with these control characters, data characters in the range FA_{16} to FF_{16} will each be expanded to a two-character string as follows:

Data Character	String Expansion
FF	FA, FF
FE	FA, FE
FD	FA, FD
FC	FA, FC
FB	FA, FB
FA	FA, 00

The end-of-file character will always mark the end of the physical record in which it is contained.

5.7.1.2 Sequential File Format. The first byte of each sequential file physical record contains the file number (positional directory entry, the same as in byte 1 of the FCB, and the second byte contains the sequential physical record number, modulo 256. The logical records, as previously described, occupy the remaining space in the physical record. The logical records may span physical record boundaries and will be stored in contiguous sectors. There can be multiple end-of-file records in a sequential file.

5.7.2 RELATIVE RECORD FILE

5.7.2.1 Record Format. Relative record logical records are of fixed length. The system uses as many physical records as necessary to contain a logical record. The maximum size for a logical record is one allocation unit. There are no control characters added to the data by the operating system.

5.7.2.2 Relative Record File Format. All logical records start at the beginning of a physical record. Unused space within a physical record is wasted. There is no end-of-file character. The write pointer in the FCB is the only end-of-file indicator. Space is allocated in the file for all records from 0 to the last logical record written.

5.7.2.3 **Program File Format.** The TX990 program file is a relative record file with a record size of 256 bytes. The file is similar to the DX10 program file, with the constraint that only one program may be installed on a TX990 program file; that is, there is only one task, with its overlays, and possibly one procedure. The number of overlays is limited to 255.

The TX990 program file is created by the link editor in image format. (See the *Model 990 Computer Link Editor Reference Manual,* part number 949617-9701.) The file has the format shown in figure 5-18. Record zero is an overhead record, whose format is shown in figure 5-19. Records 1 and 2 form the program file directory and are described in subsequent paragraphs. The remaining records contain task, procedure or overlay images. Note that each segment image must start on a record boundary.

The layout of the program file overhead record is:

Bytes 0-1 – Must be zero. Indicates a program file.

Bytes 2-15 – Reserved. Zero fill.

Bytes 16-19 – Relative record number of the next available record in the image area.

Bytes 20-51 – ID bit map for memory-resident tasks. Unused in TX990.

Bytes 52-83 – ID bit map for memory-resident procedures. Unused in TX990.

Bytes 84-115 – ID bit map for tasks. TX990 only uses task number 1.

Bytes 116-147 – ID bit map for procedures. TX990 only uses procedure number 1.

Bytes 148-179 – ID bit map for nonreplicative tasks. Unused in TX990.

Bytes 180-211 – ID bit map for all overlays.

Byte 212 – Maximum number of tasks. Must be one for TX990.

Byte 213 – Offset for tasks. Unused in TX990.

Bytes 214-215 – Record number for start of task directory.

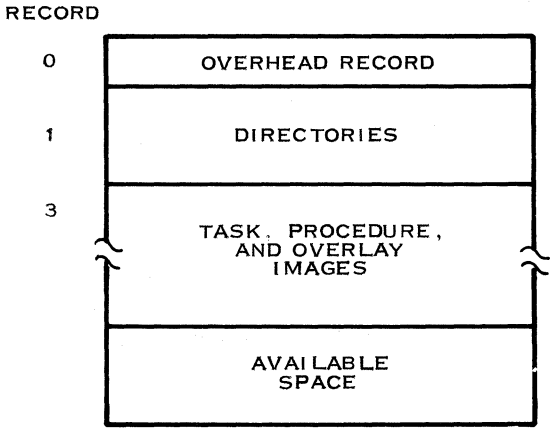


Figure 5-18. Program File Format

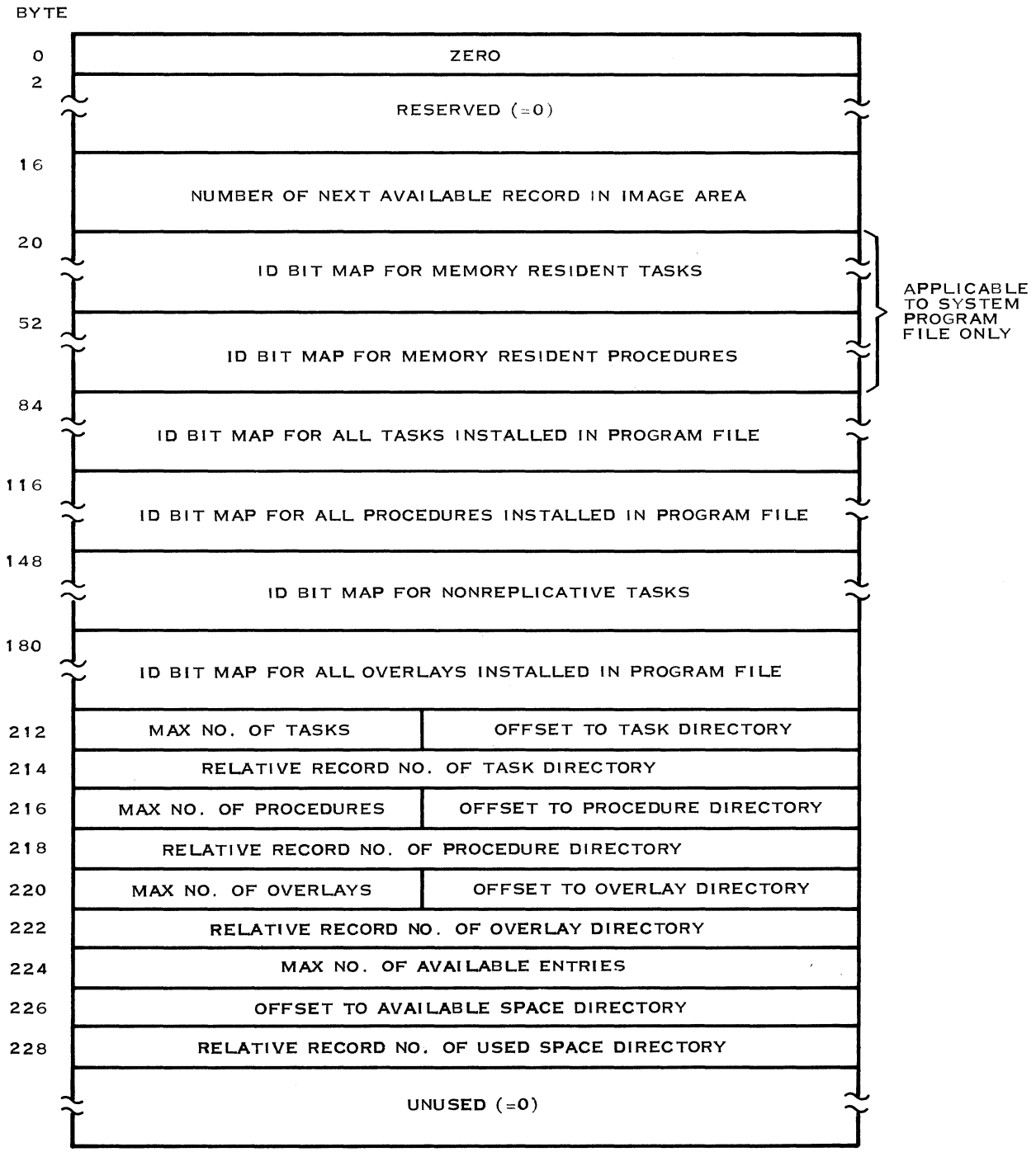


Figure 5-19. Program File Overhead File

Byte 216 – Maximum number of procedures. Must be one for TX990.

Byte 217 – Offset for procedures. Unused in TX990.

Bytes 218-219 – Record number for start of procedure directory.

Byte 220 – Maximum number of overlays.

Byte 221 – Offset for overlays.

Bytes 222-223 – Record number for start of overlay directory.

Bytes 224-225 – Maximum number of holes in file.

Bytes 226-227 – Offset for start of space directory.

Bytes 228-229 – Record number for start of unused space directory.

Bytes 230-255 – Unused. Set to zeros.

The layout of the directory records is shown in figure 5-20. Data in these fields are varying lengths, depending on the number of tasks, procedures, and overlays in the program file. All may be one or more records.

Bytes 0-(A-1) – Names of the tasks in the file. Eight ASCII bytes per name.

Bytes A-(B-1) – Directory entries for tasks. Sixteen bytes per task. See figure 5-21A.

Bytes B-(C-1) – Names of the procedures in the file. Eight ASCII bytes per name.

Bytes C-(D-1) – Directory entries for procedures. Sixteen bytes per procedure. See figure 5-21B.

Bytes D-(E-1) – Names of the overlays. Eight ASCII bytes per name.

Bytes E-(F-1) – Directory entries for overlays. Sixteen bytes per overlay. See figure 5-21C.

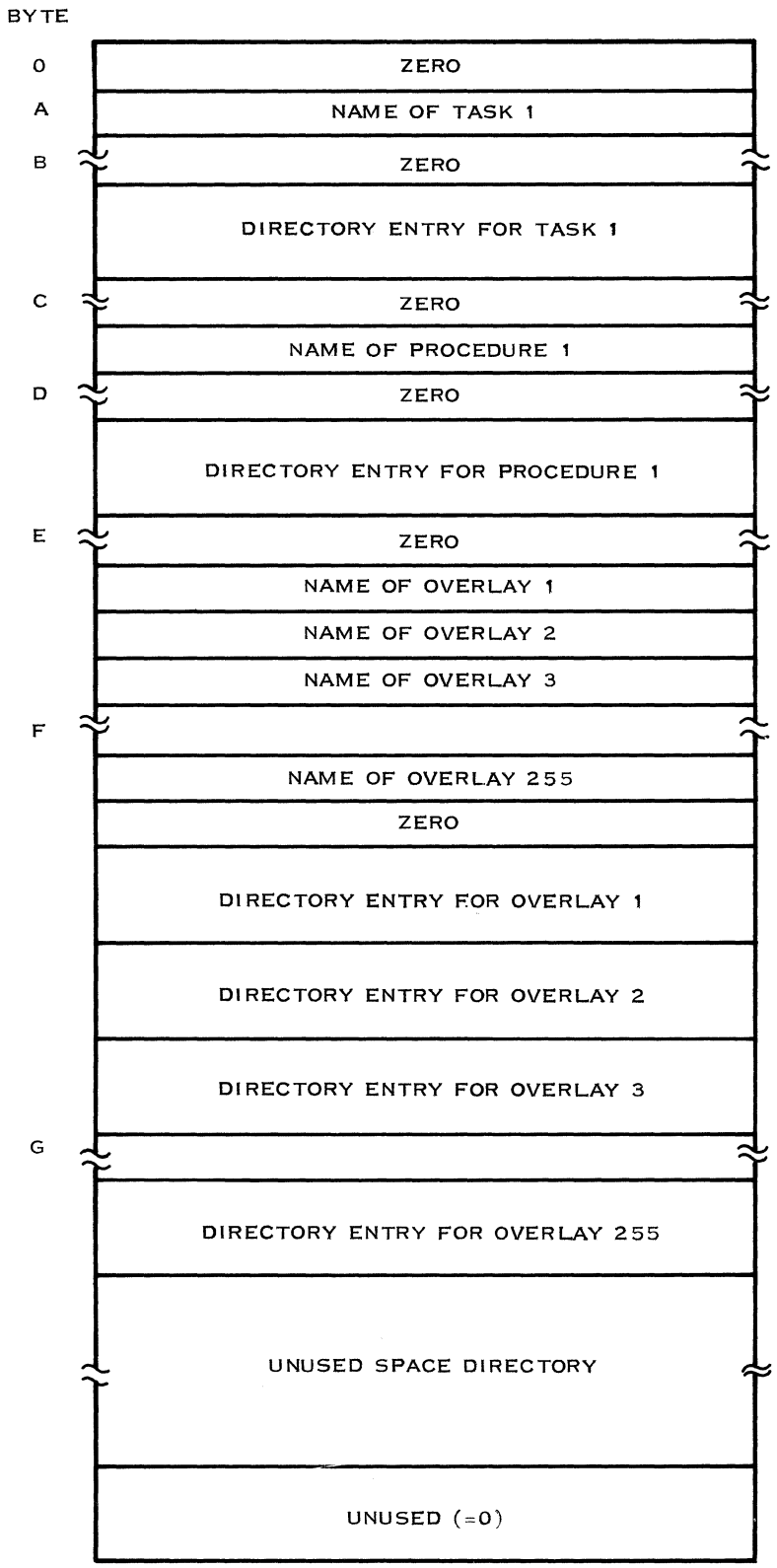
Bytes F-(F+1) – Number of unused directory entries in the file.

Bytes (F+2) - (F+3) – Number of records in the file.

Bytes (F+4) - (F+5) – Number of records available in file.

Bytes (F+6) - (G-1) – Used space directory. Four bytes per entry. An entry for each task, procedure, and overlay. Unused by TX990.

The next record is the first record of the image data. Each module is followed by a relocation bit map.



(A)137513

Figure 5-20. Program File Directory

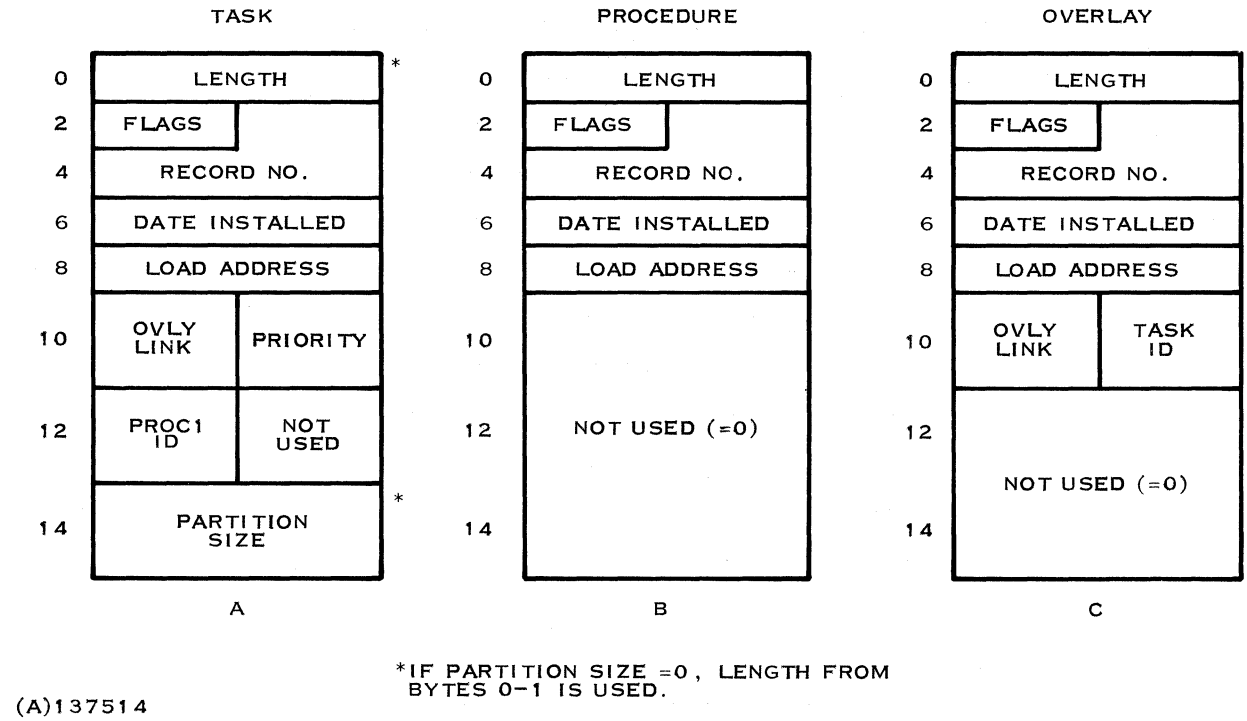


Figure 5-21. Directory Entry Format

The task directory entry data is:

Bytes 0-1 – Length of the task segment in bytes.

Byte 2 – Flags. When set indicate:

Bit 0 Privileged.

Bit 1 System.

Bit 2 Memory resident.

Bit 3 Delete protected.

Bit 4 Replicative.

Bits 5-7 Unused.

Bytes 3-5 – Record number. File record number of the start of the task image.

Bytes 6-7 – Date installed. Date is in the format:

Bits 0-6 Year displacement from 1900.

Bits 7-15 Date.

Bytes 8-9 – Load address.

Byte 10 – Overlay link. The overlay number of the first overlay associated with the task. Each overlay directory entry is in turn linked to the next entry so tasks can be associated with overlays when status or delete commands are executed. A value of 0 is used to terminate the list.

Byte 11 – Task priority.

Byte 12 – Procedure 1 ID.

Byte 13 – Not used.

Bytes 14-15 – Partition size is the size of the initial allocation for the task and overlays.

The procedure directory entry data is:

Bytes 0-1 – Length of the procedure segment in bytes.

Byte 2 – Flags. When set indicate:

Bits 0-1 – Not used (=0).

Bit 2 – Memory resident.

Bit 3 – Delete protected.

Bits 4-6 – Not used (=0).

Bit 7 – Directory entry in use.

Bytes 3-5 – Record number (24 bits). Logical record number of the start of the procedure image.

Bytes 6-7 – Date installed.

Bytes 8-9 – Load address. Must be on a beet boundary.

Bytes 10-15 – Not used (=0).

The overlay directory entry data is:

Bytes 0-1 – Length of the overlay segment in bytes.

Byte 2 – Flags. When set indicate:

Bit 0 – Relocation map is present.

Bits 1-2 – Not used (=0).

Bit 3 – Delete protected.

Bits 4-6 – Not used (=0).

Bit 7 – Directory entry is used.

Bytes 3-5 – Record number (24 bits). Logical record number of the start of the overlay image.

Bytes 6-7 – Date installed.

Bytes 8-9 – Load address. Must be even.

Byte 10 – Overlay link to next overlay.

Byte 11 – Task ID of associated task.

Bytes 12-15 – Not used (=0).

5.7.3 **FILE CONTROL BLOCK (FCB).** The format of the FCB, shown in figure 5-22, is as follows.

Bytes 0-1 – The file number (positional entry of the directory). When a file is deleted, this number is replaced with $FFFF_{16}$ to indicate that this FCB is no longer valid.

Bytes 2-5 – "FCB*" sequence, used to verify that the record is an FCB and to enable the user to easily find an FCB when doing a diskette dump.

Bytes 6-7 – Reserved for future expansion.

Bytes 8-9 – Relative record length, the number of bytes in a logical record for a relative record file. This field is zero for a sequential file.

Bytes 10-13 – Write pointer. For a sequential file, the write pointer is the end-of-media (i.e. the last record written in the last sub-file). The end-of-media is updated after each sequential write. Any record past end-of-media cannot be accessed. Bytes 10-11 contain the AU address of the end-of-media relative to the beginning of the file. Bytes 12-13 are the physical record within the AU of the end-of-media.

For a relative-record file, the write pointer is the address of the end-of-file record. End-of-file for a relative-record file is either the logical record where an end-of-file has been written or the largest-numbered logical record written, whichever occurred last. Bytes 10-13 are a 32-bit record number.

Bytes 14-17 – Current pointer, only used for sequential files. This field has no meaning for relative-record files. The current pointer is used to maintain the current read or write position within a sequential file. It is the address of the next record to be read or written. Bytes 14-15 and bytes 16-17 are the AU record address and physical record address, respectively, of the next physical record to be read or written.

Bytes 18-19 – Current byte pointer, a pointer to the beginning byte address within a physical record of the next logical record to be read or written. This field has no significance for relative record files. When the last operation before a file is closed is a write operation, the byte pointer is always set to 2, the first available byte of the next physical record. This is done to fill out the current physical record so the blocking buffer may be written to disc and released. If a read, backspace, or forward space occurred before closing the file, the byte pointer may be other than 2. In this case, the byte pointer points into the physical record preceding the one indicated to be the current physical record pointer by bytes 14-17.

The rest of the FCB, beginning at byte 20, consists of 4-byte entries defining the blocks of allocation units allocated to the file. The first two bytes are the AU address of the first allocation unit in the block. The second two bytes are the number of contiguous allocation units in the block. An $FFFF_{16}$ in the first two bytes of an entry defines previous entry as the last current entry of the FCB.

When a file is open, a copy of the FCB is maintained in memory. Some of the fields have a different meaning when in memory than when on the disc. These differences are noted in figure 5-22.

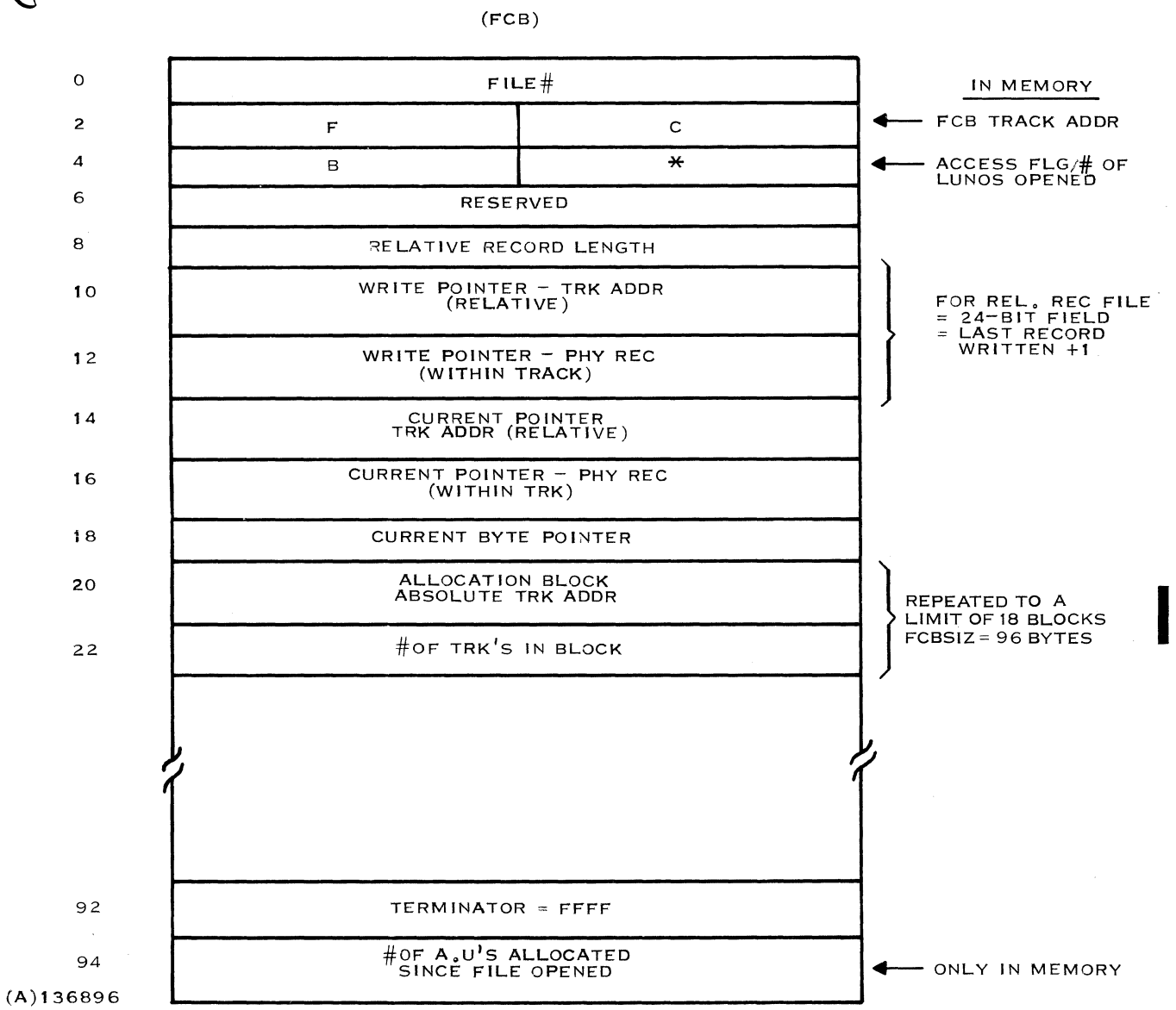


Figure 5-22. File Control Block (FCB)

SECTION VI

MODULE DESCRIPTIONS

6.1 GENERAL

This section provides a brief description of the function of each object module that may be included in the system. This description enables the user to determine the correct modules needed for a particular application or enables the system programmer to locate the modules which relate to the particular programming exercise.

6.2 TX990 KERNAL MODULES

6.2.1 **TXROOT.** TXROOT contains data and routines required for the TX990 system. Each routine is described in table 6-1.

Table 6-1. TXROOT Routines

Name	Meaning
Data Structures	The data structures contained in TXROOT are those that are configuration independent. These structures include queue headers, supervisor call table, and flags. See also the paragraphs on TXROOT in the preceding section.
Scheduler	Handles scheduling and execution of tasks.
Device/File Close Logic	Closes files or devices assigned to a task that is terminating.
End Task/End Program Supervisor Call	SVC processor for end task and end program SVCs.
Interrupt and XOP Return Points	Common return area for interrupt and XOP processors returning control to the operating system.
Power Fail/Restart	Code executed when a power fail occurs, or when power is restored.
I/O Initialization Logic	Initializes VDTs and PDTs after the operating system is loaded or during manual restart.
Supervisor Call Processor	Decodes XOP 15 and calls appropriate processor.
Fatal Error Logic	Entered when a CPU error is detected, or an illegal XOP or interrupt occurs.
Clock Interrupt Handler	Executed on each clock interrupt. It sets the flag after the specified number of clock interrupts to indicate that a system-time interval has expired.

Table 6-1. TXROOT Routines (Continued)

Name	Meaning
Queue and Dequeue Routines	Routines to handle queueing and dequeueing of TSBs.
Unsuspend Time Delay Task	Unsuspends tasks that are in time delays when their delay time has expired.
Update Time and Day	Maintains system time and day counters.
Bid Task	Places tasks on the active queue to be executed.

6.2.2 **TSKFUN.** TSKFUN contains a number of supervisor call processors for controlling task execution. The individual routines are described in table 6-2.

Table 6-2. TSKFUN Routines

Name	Meaning
Bid Task SVC Processor	This routine calls the bid task routine in TXROOT and places the previous task state in the SVC block.
Get Parameters SVC Processor	Moves task parameters from TSB to SVC block.
Get Own I.D. SVC Processor	Return I.D. of calling task in SVC block.
Get System Table Address SVC Processor	Places system table address in SVC block.
Make Task Privileged SVC Processor	Sets privileged bit in task's status register.
Do Not Suspend SVC Processor	Sets the value of the Do Not Suspend timer.
Time Delay SVC Processor	Sets task state to time delay and initializes the delay counter.
Activate Time Delay Task SVC Processor	Reactivates time delay task by clearing delay counter.
Change Priority SVC Processor	Changes priority of task to priority specified in SVC block.
Unconditional Wait SVC Processor	Task state changed to unconditional wait.
Activate Waiting Task SVC Processor	The specified waiting task is reactivated.
Get Time and Date SVC Processor	The values of year, day, hour, minute, and second are placed in the SVC block.
Initialize Date and Time SVC Processor	The values of year, day, hour, minute, and second in the SVC block are used to initialize the system timer.
Get Task Common SVC Processor	Returns address of common in SVC block.

6.2.3 **IOSUPR.** IOSUPR contains a number of routines used to process the SVC calls with code zero. The individual routines are described in table 6-3.

Table 6-3. IOSUPR Routines

Name	Meaning
I/O Call Processor	All zero code SVC calls enter here. This routine is the main driver for the I/O system. Other routines within IOSUPR are called for validation of LUNO, operation code, etc. File management, file utility, and the DSRs are called for further processing.
I/O Queueing Logic	When the desired PDT is busy, the TSB of the task issuing the I/O call is queued on the PDT.
Dummy Device I/O	Routine called if I/O is to dummy.
Error Exits	A group of entry points are provided to process various I/O errors.
Process File I/O Call	Sets up call and queues TSB for file management.
Process File Utility I/O Call	Sets up call and queues TSB for file utility routine.
GETDEV	Maps LDT and PDT to LUNO in I/O call.
FNDLDT	Maps LUNO to LDT.
BZYCHK	Routine checks whether or not the PDT is busy. If busy, the error flag and code in the PRB are set.
SETWPS	Routine called by DSR to restore return context.
GTDVNM	This routine gets the internal device number for the specified LDT.
CASTYP	This routine is called from DSR733 to place the cassette device type in the calling block.
CKCALL	This routine validates I/O requests and performs open/close housekeeping.
WIOC	This is the SVC processor for the wait I/O supervisor call.
ABTPRB, ABRTIO	This routine is the SVC processor for the abort SCB and abort I/O supervisor calls.
ABTPDT	This routine aborts the I/O for a specified PDT.

Table 6-3. IOSUPR Routines (Continued)

Name	Meaning
ENDRCD	This routine is called by a DSR when an I/O operation has been completed and end-of-record processing is required. The end-of-record routine closes the device, if it is record oriented, by clearing the TSB in the PDT, clears the busy flag in SCB and PDT, and reactivates the task if it is suspended on I/O.
TIMOUT	This routine is entered every system interval to check if a DSR is to be reentered or timed out.

6.2.4 **CNVRSN.** This module contains the four numeric conversion supervisor call processors.

6.2.5 **MEMSVC.** This module contains the supervisor call processors for Get Memory and Return Memory in the single dynamic task system.

6.2.6 **TBUFMG.** This module contains the buffer management routines. See Section V for a description of buffer management and the buffer pool.

6.2.7 **TSKLDR.** This module is the system task loader routine (see Section II) in the single dynamic task system.

6.2.8 **TITTCM.** This module contains the SVC processors for the intertask communications supervisor calls, Get Data and Put Data.

6.2.9 **CRTPRO.** This module preprocesses the Get Character, conditional Get Character, and 911 and 913 VDT utility SVC calls. This routine determines the device type of the station number in the SVC call block and calls the appropriate processor.

6.2.10 **STA913.** This module contains the keyboard interrupt handler for the 913 VDT and SVC processors for the Get Character SVC and the conditional Get Character SVC. CRTPRO must be included in a system containing STA913.

6.2.11 **STA911.** This module contains the keyboard interrupt handler for the 911 VDT and SVC processors for the Get Character SVC and the conditional Get Character SVC. CRTPRO must be included in a system containing STA911.

6.2.12 **SVC913.** This module is the SVC processor for the 913 VDT utility SVC call. CRTPRO must be contained in a system containing SVC913.

6.2.13 **SVC911.** This module is the SVC processor for the 911 VDT utility SVC call. CRTPRO must be included in a system containing SVC911.

6.2.14 **EVENTK.** This module contains the routines required to support the break key facility in TX990.

6.2.15 **DTASK.** This is the diagnostic task (see Section II).

6.2.16 **TXSTRT.** This routine contains the initialization code that is executed at initial program load, manual restart or power restart. This module may be placed after TXEND in the link causing it to be overlayed after initialization, conserving memory and eliminating the restart capability.

6.2.17 **TXEND.** This module contains external definitions for entry points which can be optionally left out of a given TX990 system. TXEND also contains the initial system entry point after loading. It initializes the interrupt vectors and XOP traps, and determines the amount of memory present in the system. It then enters TXSTRT to complete the system initialization. All modules linked after this module will be overlayed as part of the dynamic task area.

6.2.18 **STASK.** This is the start task which prints the TX990 header message when the operating system is first loaded into memory. This task is usually overlayed by the dynamic task area (see Section II).

6.2.19 **IMGLDR.** This module contains the routine to load programs from program files.

6.2.20 **DMEMSVC.** This module contains the supervisor call processors for Get Memory and Return Memory supervisor calls in the multiple dynamic task system.

6.2.21 **DYNTSK.** This module contains the logic to build and delete the data structures for the installation and deletion of task and procedures in the multiple dynamic task system.

6.2.22 **DTSKLDR.** This module contains the system task loader in the multiple dynamic task system.

6.3 DEVICE SERVICE ROUTINES

6.3.1 **FPYDSR.** This is the DSR for the diskette. The user may access a file on the diskette with the standard supervisor call block (SCB). To access the diskette directly requires a direct disc SCB (see Section III).

6.3.2 **DSR733.** This is the DSR for the 733 ASR. This DSR will handle I/O to both keyboard/printer and cassettes. DSR733 may also be used with the 743 KSR or the 820 KSR.

6.3.3 **KSRDSR.** This is the DSR for the 733 KSR, 743 KSR, or 820 KSR. This DSR will handle I/O to the keyboard/printer only.

6.3.4 **DSR913.** This is the DSR for the 913 VDT. The module STA913 must be included in the system if DSR913 is included. A system may contain both DSR913 and DSR911.

6.3.5 **DSR911.** This is the DSR for the 911 VDT. The module STA911 must be included in the system if DSR911 is included. A system may contain both DSR913 and DSR911.

6.3.6 **LPDSR.** This is the DSR for the serial interface line printer. The user may specify write ASCII or write direct. Write ASCII allows only ASCII characters and some control characters to pass to the printer. Write direct does not filter any characters from the line printer. Write direct should be used to make use of the special control features available on the TI line printer (model 810 line printer).

6.3.7 **CRDSR.** This is the DSR for the card reader.

6.3.8 **DSRTTY.** This is the DSR for the 33 ASR.

6.3.9 **DSR5MT.** This is the DSR for the TI 5-MT digital I/O subsystem.

6.3.10 **DIGDSR.** This is the DSR for the 32-input transition detection module.

6.3.11 **FLPDSR.** This is the DSR for parallel interface line printers (Models 2230 and 2260).

6.3.12 **DSR979.** This is the DSR for the Model 797A Magnetic Tape Unit.

6.3.13 **ASR9902.** This is the DSR which supports I/O to the 733 ASR/KSR and the 820 KSR through the 9902 or 9903 communications ports on the Model 990/5 computer.

6.3.14 **KSR9902.** This DSR supports I/O to the 733 or 820 KSR through the 9902 or 9903 communications ports on the Model 990/5 computer.

6.3.15 **LP9902.** This DSR supports I/O to the serial interface line printer through the 9902 or 9903 communications ports on the Model 990/5 computer.

6.4 FILE MANAGEMENT

File management consists of a number of modules that are grouped into a reentrant procedure (FMP), a task for each disc drive supported by the system, and file utility routine task (FUR). If files are to be supported, all of file management must be included in the system.

6.4.1 **FILE I/O SUPERVISOR CALL PROCESSOR MODULES.** The file I/O SVC processor modules are described in table 6-4.

6.4.1.1 **TXFMP1.** This module is the data module for drive 1. This module contains the three-word task vector for the task associated with drive 1 and should be included in any disc system.

6.4.1.2 **TXFMP2.** This module is the data module for drive 2. It contains the three-word vector for the task associated with drive 2, and should be included in a two-, three-, or four-drive system.

6.4.1.3 **TXFMP3.** This module is the data module for drive 3. It contains the three-word vector for the task associated with drive 3, and should be included in a three-or four-drive system.

6.4.1.4 **TXFMP4.** This module is the data module for drive 4. This module, which contains the three-word vector for the task associated with drive 4, should be included in a four-drive system.

6.4.1.5 **TXFMP.** This is the main driver of the file management procedure. This module decodes the opcodes and calls the appropriate opcode processor.

6.4.1.6 **FMOPEN.** This module contains the open, open rewind, and rewind file processors. The file is opened, and when needed, the file utility routine is called to auto-create the file.

6.4.1.7 **FMCLOS.** This module contains the processors for close, close-unload, and close-EOF file. This module also contains a routine to deallocate unused allocation units to the system. The de-allocation routine is also called by the FUR compress routine.

6.4.1.8 **FMREAD.** This module contains the read file processor. This module issues reads to the DSR, performs the unblocking of physical records, and converts encoded control characters to their decoded values.

6.4.1.9 **FMWRIT.** This module contains the write file processor. This module performs the blocking of logical records, encoding of certain control characters, and calls the DSR to write the physical record.

6.4.1.10 **FMFBSP.** This module contains the forward and backspace routines for diskette files.

Table 6-4. File I/O SVC Processor Modules

Name	Meaning
TXFMP1	This module is the data module for drive 1. This module contains the three-word task vector for the task associated with drive 1 and should be included in any disc system.
TXFMP2	This module is the data module for drive 2. It contans the three-word vector for the task associated with drive 2, and should be included in a two-, three-, or four-drive system.
TXFMP3	This module is the data module for drive 3. It contains the three-word vector for the task associated with drive 3, and should be included in a three- or four-drive system.
TXFMP4	This module is the data module for drive 4. This module, which contains the three-word vector for the task associated with drive 4, should be included in a four-drive system.
TXFMP	This is the main driver of the file management procedure. This module decodes the opcodes and calls the appropriate opcode processor.
FMOPEN	This module contains the open, open rewind, and rewind file processors. The file is opened, and when needed, the file utility routine is called to auto-create the file.
FMCLOS	This routine contains the processors for close, close-unload, and close-EOF file. This module also contains a routine to deallocate unused allocation units to the system. The deallocation routine is also called by the FUR compress routine.
FMREAD	This module contains the read file processor. This module issues reads to the DSR, performs the unblocking of physical records, and converts encoded control characters to their decoded values.
FMWRIT	This module contains the write file processor. This module performs the blocking of logical records, encoding of certain control characters, and calls the DSR to write the physical record.
FMFBSP	This module contains the forward and backspace routines for diskette files.
FMUTIL	This routine contains a number of routines that are called by the other file management routines. The individual routines are described below:
GETTSB	Gets next TSB from file management queue.
FNDTRM	Finds end-of-file control block (FCB) in memory. FUR also calls this routine.
SEMAPH	Routine to gain exclusive access to a resource. FUR also calls this routine.
GETTRK	Maps allocation unit and record number address relative to beginning of file into physical allocation unit and record.

Table 6-4. File I/O Routines

Name	Meaning
FMUTIL	This routine contains a number of routines that are called by the other file management routines. The individual routines are described below:
GETTSB	Gets next TSB from file management queue.
FNDTRM	Finds end-of-file control block (FCB) in memory. FUR also calls this routine.
SEMAPH	Routine to gain exclusive access to a resource. FUR also calls this routine.
GETTRK	Maps allocation unit and record number address relative to beginning of file into physical allocation unit and record.
NXTREC	Updates current pointers in file control block.
GETCUR	Maps relative record number of relative-record file into allocation and record address relative to start of file.
RLOCK	Checks if relative-record file record is locked.
VALLDT	Verifies that LUNO has not already been opened by another task with exclusive access and handles linkage to FCB in memory for shared files.
WRTFCB	Writes file control block to the disc. FUR also calls this routine.

6.4.2 **FILE UTILITY MODULE DESCRIPTIONS.** The following modules make up the file utility task (FUR, task $0B_{16}$), (see table 6-5). When a file I/O task (FMP1, FMP2, FMP3, or FMP4) is included in the system, all of the file utility modules must be included. When only assign and release LUNO operations are desired and file management is not desired, the module FURSVC must be included without the rest of the file utility modules.

6.4.2.1 **FURTSK (File Utility Task).** Removes tasks from the file utility queue and decodes the operation code, then branches to the operation processor. When the operation processor completes its task, it returns to FURTSK. FURTSK places the calling task back on the active queue and continues to remove tasks from the file utility queue until all requests are satisfied.

6.4.2.2 **FURSVC (Assign and Release LUNO).** This module includes support to assign and release LUNOs using the SVC code 00_{16}, opcodes 91_{16}, and 93_{16}. This modules also includes the supervisor call 15_{16} support.

6.4.2.3 **FILESVC (Create, Delete, and Compress Files).** This modules supports supervisor call code 00_{16}, opcodes 90_{16}, 92_{16}, and 94_{16}.

6.4.2.4 **CHSVC (Change File Name and Change File Protection and Check File Name Syntax).** This module supports SVC 00_{16}, operation codes 95_{16}, 96_{16}, 97_{16}, 98_{16}, and 99_{16}.

6.4.2.5 **ALUNIT (Allocate and Deallocate Allocation Units).** This module is used by file I/O processors and by file utility processors. It allocates and deallocates allocation units.

Table 6-5. File Utility Routines

Name	Meaning
ALSVC	Assigns a LUNO to a device or file.
RLSVC	Release a LUNO from a device or file.
ALPHA/ALPNUM	Check range of a character.
DNTPDT	Search device name table (DNT) for a device name.
FLDT	Build a file LDT.
SYNX	Check syntax of a file name and put it into an LDT block.
LDTOK	Validate that a LUNO is not already assigned.
ALUNIT	Allocate diskette allocation units.
RLUNIT	Deallocate diskette allocation units.
FNDSP	Find contiguious blocks of allocation units.
ALOSP	Set the allocated allocation units bits "on" in the allocation bit map on the diskette.
SERDIR (Search the Directory)	This module is used by file I/O processors and by file utility processors. It earches the directory for a diskette file name.
CRESVC	Builds an LDT-like structure with the file pathname.
CREHRD	Physically create the file on the diskette.
RDINF	Reads the disc control block (DCB) and initializes the direct disc PRB.
WRDIR	Writes the directory entry to the diskette.
DFSVC	Delete file SVC processor.
RDFCB	Read the file control block (FCB) and verify that it is a file control block.
RLTRK	Deallocate tracks in a file control block (FCB).
SETUP	Builds an LDT structure, and searches the LDT list to see if the file is already opened. Then searches the directory for that file.
CMSVC	Compresses a file by trimming off the excess allocation units. Calls the I/O routines in FMUTIL.

Table 6-5. File Utility Routines (Continued)

Name	Meaning
NMSVC	Change the name SVC processor.
USVC	Unprotect the file SVC processor.
WSVC	Write protect the file SVC processor.
DSVC	Delete protect the file SVC processor.
SEARCH	Set up the direct disc PRB and search the directory.
BLD	Build an LDT structure for devices or files.
SYNSVC	Check the syntax of a file name SVC processor.

6.4.3 VOLUME (VOLUME NAME SUPPORT). The volume task is a file management task that locates volume names (table 6-6). When a LUNO assigned to a file using volume names is opened, the volume task reads from each disc drive searching for the volume name. When the volume name is located, the file management ID is placed in the LDT file management task associated with that drive bid.

When volume name support is desired, the module volume must be included when the system is being generated. When volume name support is not included in the system and volume names are used in supervisor calls, an error >21, indicating a bad disc name, is returned to the calling task.

Table 6-6. Volume Modules

Name	Meaning
VOLNAM	Whenever a file is being opened by a task, VOLNAM puts the task on the VOLQUE, then bids VOLTSK. Called by IOSUPR.
VOLTSK	Bid by VOLNAM. Takes tasks off VOLQUE, locates the volume in the diskette drive, and puts the task on FMPQUE and bids a file management task.
FNDVOL	Sets the semaphore FMPFLG and calls FNVOL. Returning from FNVOL, it clears the FMPFLG. Called by FURTSK modules.
FNVOLU	Calls FNVOL. Has the same calling interface as FNDVOL. Called by FURTSK modules.
FNVOL	Scans the PDT list until a disc PDT is found. Then it reads that diskette and compares the LDT's volume name to the diskette's volume name. When they compare, the file management ID is placed in the LDT.

6.5 OPERATOR COMMUNICATION PACKAGE (OCP)

OCP consists of a number of modules which provide the operator with a means of communicating with the operating system. Four modules are required when OCP is included in the system; the rest can be optionally included. Each of the optional modules contains a number of individual command processors.

6.5.1 **OCPTSK.** This module contains the data base for OCP and is a required module.

6.5.2 **OCPTBL.** This module contains the entry points for each of the individual OCP command processors. This module is required.

6.5.3 **OCPPRC.** This module is the main driver for OCP and is required. This module contains a number of individual routines which are described in table 6-7.

Table 6-7. OCPPRC Routines

Name	Meaning
OCP Read	Reads command line.
Input Record Pre-Scanner	Reformats the command line to a specific format.
Argument Manipulation Routines	Routines used by individual processors to get parameters, convert to hex number if needed, and scan command line.
Decode and Call	Decodes next command and calls correct processor.
Common Return Routine	Provides common return and error returns for processors.
I/O Routines	Routine to handle I/O for rewind, forward space, and backspace processors.

6.5.4 **OCPLRT.** Optional processor in single dynamic task system to assign LUNO (ALUNO), release LUNO (RLUNO), execute program (EXECUTE), load non-real time program (LPROG), and load real-time program (LRPROG). The modules FURSVC and TSKLDR must be included also.

6.5.5 **DOCPLRT.** Optional processor in multiple dynamic task system to assign LUNO (ALUNO), release LUNO (RLUNO), execute program (EXECUTE), load non-real time program (LPROG), load real-time program (LRPROG), install non-real task (ITASK), install real-time task (IRTASK), install procedure (IPROC), delete task (DTASK), and delete procedure (DPROC–. The modules FURSVC, DYNTSK, and DTSKLDR must be included also.

6.5.6 **OCPSLD.** Optional processor for debug commands: dump memory (DMEM), load memory (LMEM), add (ADD), subtract (SUBTRACT), jump (JMP), set breakpoint (SBKPT), and clear breakpoint (CBKPT). The module CNVRSN must be included also.

6.5.7 **OCPIOU.** Optional processor in single dynamic task system for task status (STASK), I/O status (SIO), rewind (REWIND), backspace (BSPACE), and forward space (FSPACE).

6.5.8 **DOCPIOU.** Optional processor in multiple dynamic task system for task status (STASK), procedure status (SPROC), I/O status (SIO), rewind (REWIND), backspace (BSPACE), and forward space (FSPACE).

6.5.9 **OCPTAD.** Optional processor to install time and data (IDATE) and display time (TIME). The modules TSKFUN and CNVRSN must be included if OCPTAD is included.

6.5.10 **OCPTLD.** Optional processor for dump workspace (DWORK), kill task (KTASK), kill I/O (KIO), and trace address (TRACE).

6.5.11 **OCPEND.** This module supplies dummy command table entries to satisfy references to processors that are not included in the system. This module is required unless all optional parts are included in the system.

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 Documentation (944776-9701)

Manual Date: 1 October 1978 **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.

CUT ALONG LINE

Location in Manual	Comment/Suggestion
_____	_____
	_____
	_____
	_____
_____	_____
	_____
	_____
	_____
_____	_____
	_____
	_____
	_____

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

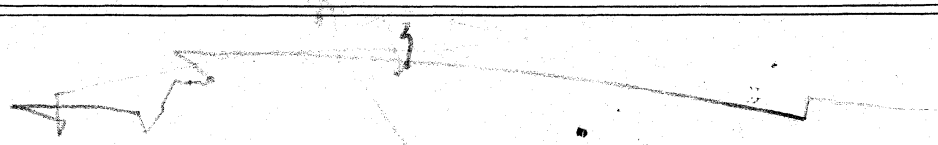
POSTAGE WILL BE PAID BY

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