Publication Number B29-434

OS/16 MT 2
PROGRAM
LOGIC MANUAL

INTERDATA®
Subsidiary of PERKIN-ELMER
Oceanport, New Jersey 07757, U.S.A.

Printed in U.S.A.
October 1976

PAGE REVISION STATUS SHEET

PUBLICATION NUMBER B29-434

TITLE OS/16 MT2 Program Logic Manual

REVISION R00 DATE 10/76

PAGE	REV.	DATE	PAGE	REV.	DATE	PAGE	REV.	DATE
1-1 to 1-2	R00	10/76						
2-1 to 2-18	R00	10/76						
3-1 to 3-6	R00	10/76						
4-1 to 4-16	R00	10/76						
5-1 to 5-34	R00	10/76						
6-1 to 6-24	R00	10/76						
7-1 to 7-18	R00	10/76						
8-1 to 8-4	R00	10/76						
A1-1 to A1-4	R00	10/76						
A2-1 to A2-14	R00	10/76						

FOREWORD

This Manual describes the OS/16 MT2 R01, specifically, and is designed as a guide to the internal structure of the operating system. Use of this Manual requires that the reader be knowledgeable of the features, functions and conventions of OS/16 from the user's point of view as documented in the following manuals:

OS/16 MT2 Programmer's Reference Manual, Publication Number Number 29-249

OS/16 MT2 Operator's Manual, Publications Number 29-430

OS/16 MT2 System Planning and Configuration Guide, Publication Number 29-431

The reader should also be familiar with the 16-Bit series architecture and its features as described in:

16-Bit Processor User's Manual, Publication Number B29-509

Chapter 1 is a general introduction to the system. Chapter 2 of this manual describes the general structure of OS/16 MT2. Chapter 3 discusses the conventions followed by the system in terms of interfacing between modules, naming of fields and flag bits, and the structure of modules. Chapter 4 contains a description of the OS overlay scheme. Chapters 5, 6 and 7 contain a detailed technical description of the major modules in OS/16. These chapters are designed to provide a technical overview of the system. Chapter 8 discusses executive tasks and user added extensions to OS/16. Appendix 1 contains a list of system crash and journal codes and their meanings. Appendix 2 contains the format of system control blocks.

Table of Contents

ILLUSTRATIONS

TABLES

APPENDICES

CHAPTER 1
INTRODUCTION

The OS/16 MT2 Program Logic Manual (PLM) is designed as a guide to the internal structure of the operating system OS/16 MT2. It is intended for use by personnel involved in maintaing and modifying the system, and is normally used in conjunction with program listings.

This manual deals exclusively and specifically with OS/16 MT2 R01. Thus, specified methods of implementation of various functions are not to be construed as the method of implementation used in all future releases of OS/16.

OS/16 MT2 is an operating system that provides an efficient and powerful means of using the resources of an INTERDATA 16-Bit Processor in a multi-tasking environment. System control by the console operator and from specified files or devices, interrupt handling, I/O servicing, and inter-task communication/control are built-in functions of OS/16. The OS/16 File Manager and Device Drivers provide a powerful set of data management services for both direct access and non-direct access devices.

OS/16 MT2 functions as a compatible subset of OS/32 MT for applications programming. Indexed and contiguous disc files are compatible between systems.

CHAPTER 2
SYSTEM STRUCTURE

2.1 INTRODUCTION

This chapter presents a general overview of the structure of OS/16 MT2 from a technical viewpoint. As illustrated in Figure 2-1, OS/16 MT2 is composed of four main modules: The Executive, Command Processor, File Manager, and Driver Library. I/O support is provided by the drivers together with major portions of the executive, thus the drivers are discussed in the context of the I/O subsystem.

There are many SYSGEN options in OS/16 MT2. Some of the features described may be deleted from the system by the user. Refer to the System Planning and Configuration Guide for details.

2.2 EXECUTIVE

The OS/16 MT2 Executive provides control over, and extensions to, an INTERDATA 16-Bit Processor, as well as providing task management facilities. All requests for OS/16 MT2 services are processed by the executive's Supervisor Call (SVC) handler. Other support, provided by the executive, includes the:

- Internal interrupt handlers
- Memory manager
- Task manager and scheduler
- I/O service handler
- Crash handler
- Clock management routines
- Intertask coordination and control routines
- Image loader

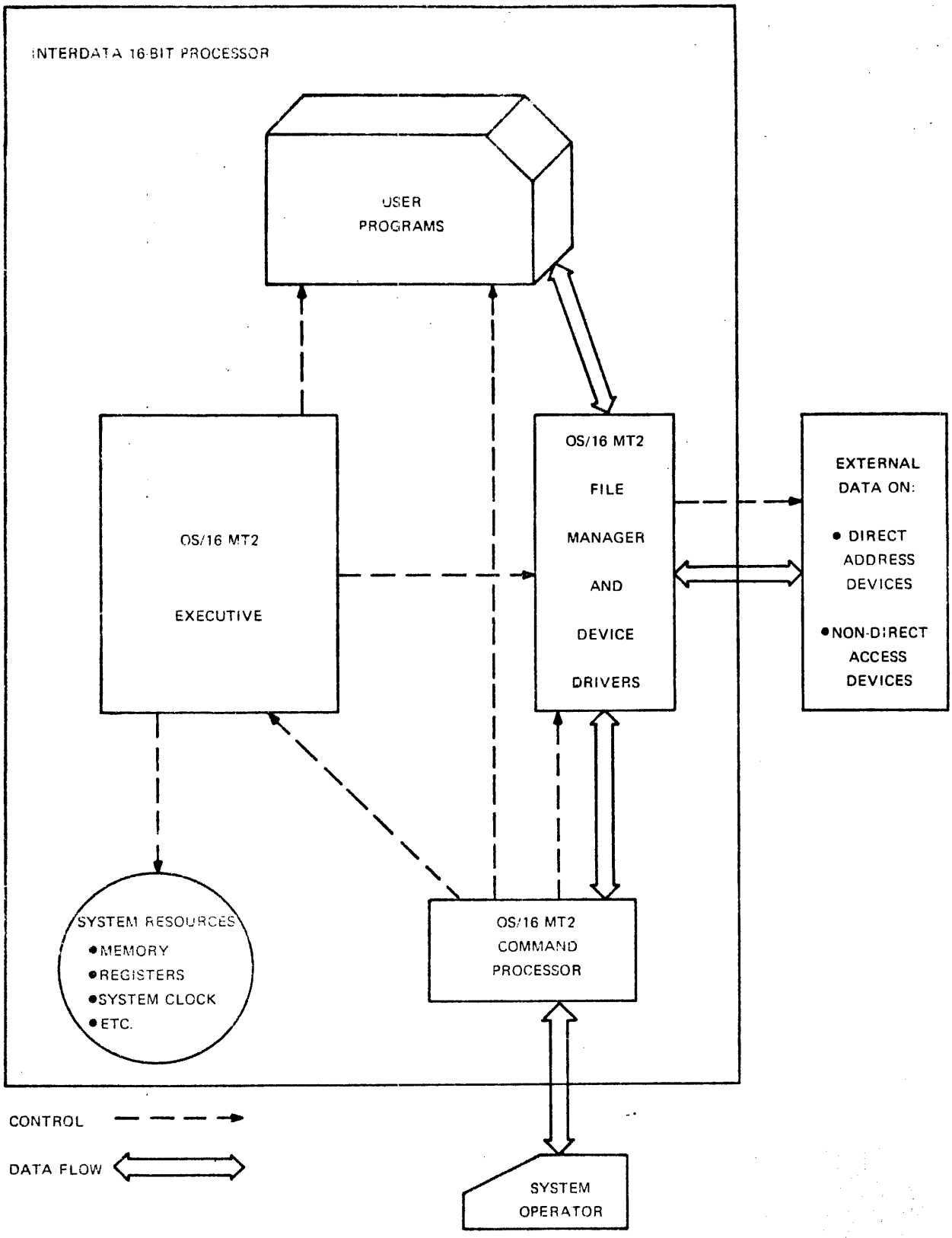


Figure 2-1 OS/16 MT2 System Overview

General utility function routines

System journal handler

2.2.1 Executive Services

SVC Handlers - For SVCs that can be overlayed, the SVC executive is entered on SVC interrupt. The SVC executive handles Roadblocking* of SVC executors The executors handled by the SVC executive are non-reentrant, but are executed in a reentrant state. Roadblocking is used to keep other tasks from entering an SVC executor that is in use.

All other SVC interrupts cause control to be passed directly to the SVC handler. These SVCs run in a non-reentrant state.

Memory Manager - Memory is divided into three areas: the operating system, task partitions, and dynamic system space. (See Figure 2-2.)

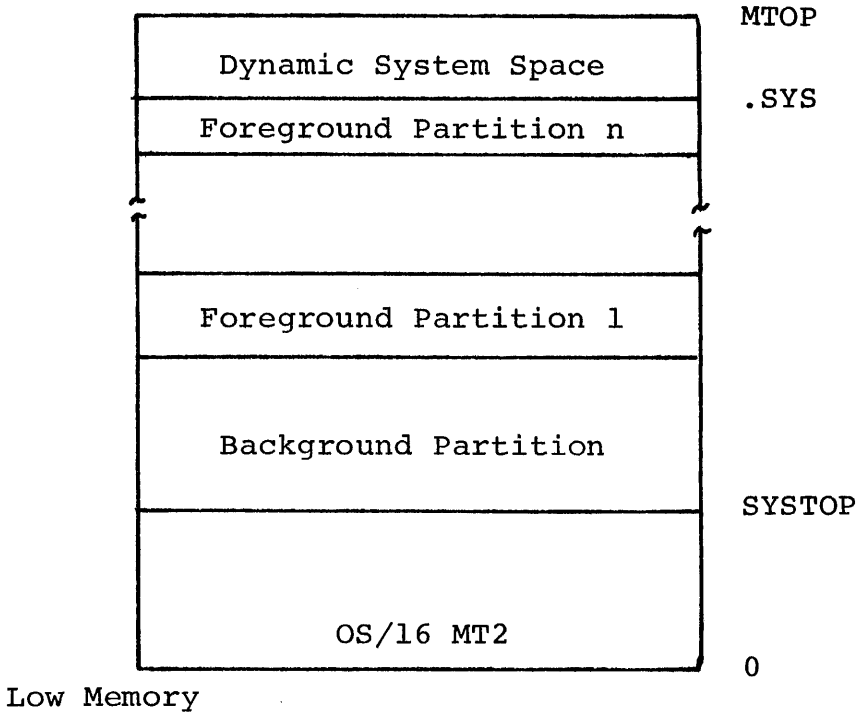


Figure 2-2 Memory Map of Entire System

*Roadblocking prevents the operating system from attempting to service two concurrent SVC requests of the same type (e.g. two SVC 2's)

2.2.2 Task Management

An OS/16 MT2 system, which includes the Command Processor module, controls a minimum of two tasks. At the system level, the command processor runs as an executive task (E-task) at the highest priority of 0. At the user level, there is at least one background partition. At system generation, the user defines the number of partitions and, therefore, the maximum number of user tasks in the system (1 to 126).

A task is controlled through a Task Control Block (TCB). A TCB is set up for every task at system generation. Each task in the system is in one of the following states: Current, Ready, Wait, Paused and Dormant.

The current task is the one executing instructions. Only one task may be in this state at any given instant in time. All other tasks in memory are in one of the other four states, but may become the current task depending on circumstances.

A ready task is one which has no obstacles to becoming the current task. It is eligible to be scheduled (i.e., become current) whenever it becomes the highest priority ready task.

A task in wait state is one which may not become ready until some specific circumstance has occurred. Among the possible wait states are:

I/O Wait	Waiting for I/O completion
Time Wait	Waiting for an interval or time of day
Trap Wait	Waiting for a task-handled trap
SVC Wait	Waiting for an SVC executor

A paused task is one which may not executive until it is explicitly

continued by the console operator. A paused task is said to be in console wait.

A dormant task is one which may not execute until it has been explicitly started, either by the console operator or by another task. When a resident task goes to End of Task (EOT), it enters the dormant state. When any task is loaded, it enters the dormant state after loading is complete, and remains in this state until it is started.

Paused and dormant are both wait states; they are listed separately since they require operator intervention.

2.2.3 Task Partitions

Memory within a partition is organized as shown in Figure 2-3. The UDL area (User Dedicated Locations) contains TSW swap areas, system pointers, and other data used for communication between the operating system and the task. Its size depends upon the type of floating point support required by the task:

UDL	X'24'	No floating point support
UDLF	X'44'	Single precision floating point
UDLD	X'84'	Double precision floating point

The main code for the task is positioned above the UDL. The first free location above the code is given by UTOP, and the last location within the partition by CTOP. The memory between UTOP and CTOP is available to the task, either directly as general workspace, or by an SVC 'Get Storage' request which dynamically updates UTOP to the next available location above the acquired 'storage' area.

In a full system, the partition boundaries are set up by the operator at run time. In a system with no command processor, they are set up at

system generation by the linking in of the tasks themselves; where the size of each task determines the size of its partition.

The system recognizes two partition types: background and foreground. A task running in the background partition is flagged by the system by setting the background bit in the task's options halfword. The background task is limited in that it may neither communicate with any other task, nor interfere in any way with the foreground tasks. A foreground partition can contain a task, reentrant library, or task common block. Support for reentrant libraries and task common block is provided through the Task Establisher (TET/16). When an established task is loaded, the resident image loader checks that, if a library and/or task common block is referenced, the relevent partition has been set up. If not, the load is rejected. At run time, the system does not distinguish between these three different types of modules.

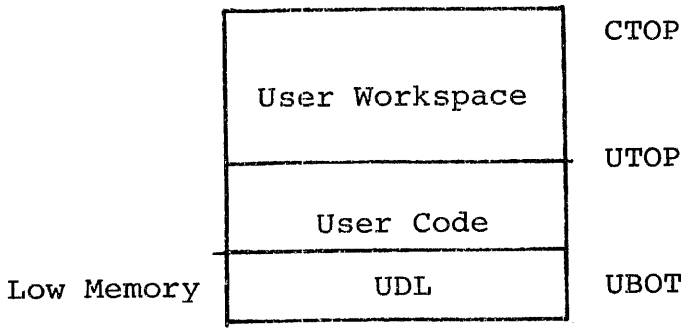


Figure 2-3 Memory Map of User Partition

2.2.4 Dynamic System Space

The dynamic system space area is available to the file manager for the temporary allocation of File Control Blocks (FCBs) on behalf of one or more tasks in the system. FCBs are positioned from the top of memory (MTOP) down; the lower limit is .SYS (see Figure 2-4).

An FCB is set up whenever a disc file is assigned and is deleted when the file is closed. The pointer FBOT and any FCBs positioned below a deleted FCB are moved up so that no gaps are left in memory. This procedure is described in Chapter 7.

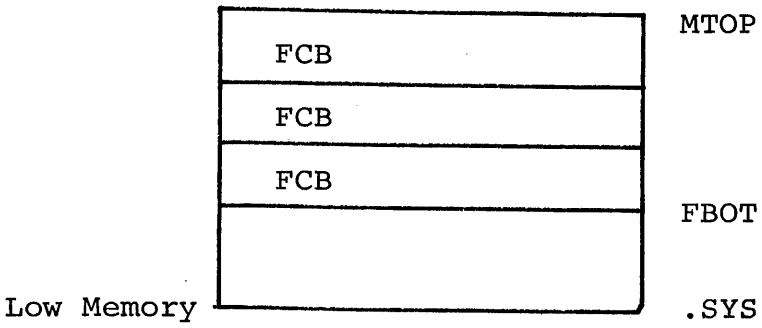


Figure 2-4 Memory Map of System Space

2.2.5 Floating Point Support

Support is provided for Single and Double Precision Floating Point (SPFP and DPFP) as a SYSGEN option. In addition, optional traps are provided for the extended SPFP instructions, available only on a Model 8/16 with SPFP hardware.

In multi-tasking systems on processors with hardware DPFP, the DPFP registers are saved during scheduling. Software traps for DPFP instructions are provided for Models 7/16 and equivalent without DPFP hardware. The registers are maintained in the task's UDL.

SPFP support is functionally similar to DPFP support described above. The registers are saved or maintained in a separate area in the UDL.

The registers save areas are reserved in the UDL as shown in Figure 2-5.

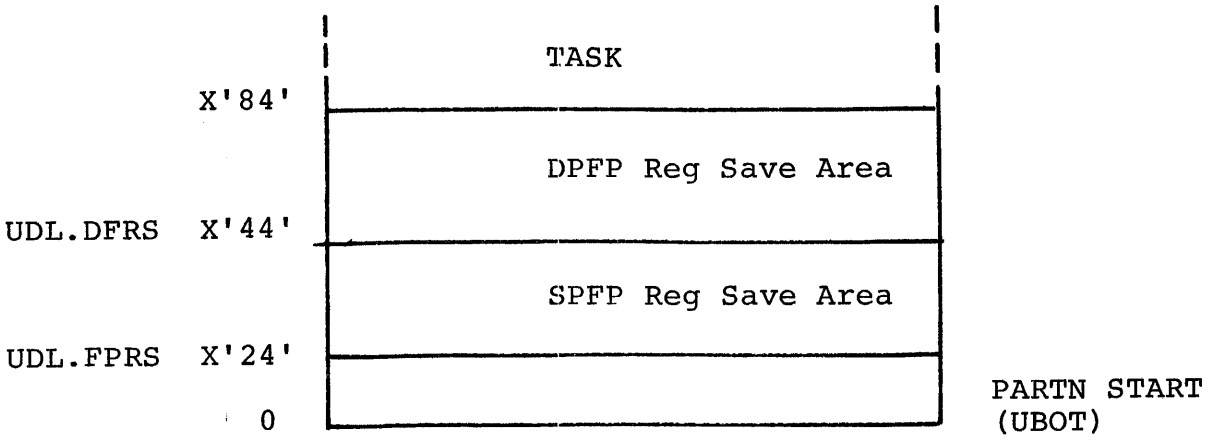


Figure 2-5 Register Save Areas in the UDL

2.2.6 Internal Interrupt Handlers

OS/16 MT2 handles the following interrupts:

Floating Point Arithmetic Fault (Processor)

Fixed Point Divide Arithmetic Fault (Processor)

Illegal Instruction (Processor)

Memory Parity Error (Processor)

Memory Protect Fault (Processor)

Illegal SVC (OS/16 MT2)

Illegal Address in SVC (OS/16 MT2)

On detection of any of these faults the default system action is to log a message on the system console and pause the current task. For some interrupts, if requested by the user, the current task continues execution rather than pausing. If the fault occurs within the operating system itself, control is passed immediately to the crash handler, where a crash code is displayed on the display panel and the processor is placed in a wait state. Note that on OS/16 MT2 systems configured with no Command Processor module, and therefore no system console device, system error messages can not be output. The task is placed in a permanently paused state.

Illegal instruction fault requires special preprocessing. Some instructions require software support, such as list instructions on a Model 7/16 Basic. Depending on options specified at system generation, the system may contain one or more of the following software traps:

List

Multiply/Divide

Single Precision Floating Point

Double Precision Floating Point

Extended Single Precision Floating Point

Control is passed to each trap routine in turn until either the instruction is recognized and can be processed, or the instruction is found to be genuinely illegal and the illegal instruction handler is entered.

2.2.7 Clock/Timing Facilities

OS/16 MT2 optionally supports both the line frequency clock and the precision interval clock. The following services are provided:

Time of day clock

Day, month, and year calendar

Interval Wait; milliseconds from now

Interval Wait; time of day

Task handled trap; milliseconds from now

Task handled trap; time of day

These services are described in Chapter 5.

2.2.8 Loader

The resident image loader loads tasks, overlays, reentrant libraries, and task common blocks that have been established into load modules

by the Task Establisher (TET/16).

TET/16 inputs loader format object code (as output by CAL or the library loader) and outputs a load module consisting of:

1. A 20 byte Loader Information Block (LIB).
2. The image of the task, overlay, etc. as a number of 256 byte records. The last record contains the exact number of bytes required to complete the task, overlay, etc.

The LIB contains all the information necessary for the module to be loaded and prepared for execution. See Appendix 2, Data Structures, for details.

2.2.9 Intertask Coordination/Communication

OS/16 MT2 provides the foreground system of tasks with a means of intertask communication and control. The features provided include:

Load a task

Start/Delay Start a task

Cancel a task

Delete a task

Queue a parameter to a task

Change a task's priority

Obtain a task's status

Send a message to a task

These functions are performed through SVC 6 calls, and the desired function is specified in the parameter block. These calls may be directed towards another task or may be self-directed.

2.2.10 Task Handled Traps

OS/16 MT2 provides the user task with a mechanism whereby it may interrupt its normal execution and enter a special subroutine upon the occurrence of certain events. The events that cause the special subroutines to be entered are:

Receipt of a parameter on user task queue

Timer completion

Power restoration

I/O proceed completion

The routine to be entered, if any, is controlled by the current value of the Task Status Word (TSW) and by the contents of the task's User Dedicated Locations (UDL). The TSW is a mask containing bits which are interpreted by OS/16 MT2 to enable (if set) or disable (if reset) the interrupt condition associated with the bit. The UDL contains addresses of special subroutines to be entered upon occurrence of an enabled event. The UDL also provides a storage area into which the value of the user's TSW and the location counter previous to the event may be saved so that the user may resume normal execution after completion of a special event handler.

The value of the TSW is manipulated through a Supervisor Call instruction (SVC 9). The addresses of subroutines, and the user status to be set when entering those subroutines, may be set up by the task storing directly into its UDL or by a pre-assembled UDL.

2.2.11 Crash Handler

This routine is entered when the system cannot continue without the risk of destroying system or user information. A crash code is displayed on the display panel and is also stored in the System Pointer

Table (SPT) at SPT.CRSH. (See Appendix 1 for crash codes and meanings.) System initialization does not reset SPT.CRSH. Some of the conditions which cause the crash handler to be entered are:

Illegal instruction within the system.

Invalid item on command processor queue.

Interrupt from device zero.

2.2.12 System Journal

The system journal is a circular list of historical data maintained by the system. Each entry on the journal consists of five halfwords of information: the task ID of the task which was active at the time of the entry, the reason for making the entry (journal code), and information pertinent to that call. The system journal is established at SYSGEN time by the Configuration Utility Program. System journal processing may be eliminated at SYSGEN time. See Appendix 1 for a list of the journal codes and their meanings.

2.3 I/O SYSTEM

The I/O System consists of system routines and control blocks necessary to provide a device independent facility for performing I/O requests. It is composed of the SVC 1 Executor, IODONE, device drivers, Device/Volume Mnemonic Tables (DMT/VMT), Device Control Blocks (DCB), Interrupt Service Pointer Table (ISPTAB), and the Logical Unit Table (LTAB).

2.3.1 Device/Volume Mnemonic Tables

All devices and direct-access volumes are referred to throughout the system either by logical unit or by an ASCII identifier. These tables, DMT and VMT, bind these ASCII identifiers to the devices' DCBs.

2.3.2 Logical Unit Table

This table is physically present in all TCBs. It is of interest to the I/O subsystem and the file manager. It consists of a table of DCB or FCB addresses, one for each logical unit. If the logical unit is not assigned to any device, the entry is set to zero. The size of the logical unit table for all user tasks is the same, and is fixed at SYSGEN time. Access privileges are placed in a byte table following the logical unit table.

2.3.3 Device Control Block (DCB)

A DCB is provided for each device in the system. This control block contains device-dependent information such as the attributes of the device, flags, and register save areas if needed. Pointers are provided to the driver initialization, interrupt service, and termination phases.

2.3.4 Interrupt Service Pointer Table (ISPTAB)

The ISP table is used to control I/O requests through the interrupt capability of the 16-Bit Series Processor.

2.3.5 SVC 1 Processor

The SVC 1 Processor saves the user's registers, picks up the user's parameter block address for the driver, and then makes several error checks. These are done primarily through the mechanism of checking the attributes bytes in the device control block against the function code specified in the call. If the call is in order, the system vectors to the appropriate driver.

2.3.6 Drivers

The initiation phase of an OS/16 Driver runs in supervisor state. The interrupt-handling phase runs with all interrupts inhibited,

except for machine malfunction. The termination phase of the driver runs in supervisor state. (See Chapter 3).

2.3.7 Trap Generating Devices

There is a class of devices, called Trap Generating Devices (TGDs), whose interrupts require the scheduling of a task to perform a service in response to that interrupt. In OS/16, the means provided to respond to these interrupts is to have the driver queue a parameter to the appropriate user task in order to respond to the event. The Instrumentation Society of America (ISA) has established standard calls for handling these devices, and OS/16 supports:

Connect a task to a TGD (Connect)

Enable interrupts from a TGD (Thaw)

Disable interrupts from a TGD (Freeze)

Disconnect a task from a TGD (Unconnect)

In addition, OS/16 supplies the user with a facility to simulate the occurrence of an interrupt from one of these devices (SINT).

2.4 COMMAND PROCESSOR

The command processor (which is also the system task) provides the operator interface to OS/16 MT2. It executes as a task in OS/16 MT2 and is designed so that many functions are performed through supervisor calls. The command processor contains routines to support the Command Substitution System (CSS), routines to do memory partitioning, and routines to support direct access devices. The command processor controls all I/O requests to the console and log devices.

2.4.1 Command Processing

The command processor accepts commands from the system console device, decodes them, and calls the appropriate executor. Some commands are executed through supervisor calls (e.g., DELETE, ASSIGN) while others are executed by the command processor routines (e.g., MARK, DISPLAY). The command processor contains logic to provide the console operator with informative messages in case of error.

2.4.2 Command Substitution System (CSS)

The Command Substitution System (CSS) routines provide the ability to build, execute, and control files of OS/16 MT2 operator commands. CSS consists of routines to execute CSS operator commands, to manage the CSS buffers, and to provide the command parameter substitution facility. The CSS buffers are established at SYSGEN time by the Configuration Utility Program.

2.4.3 Direct Access Support

The command processor provides the operator with the command functions necessary to allocate and delete files, display files, perform functions such as rewind, backspace record to a file assigned to the user task, and for mounting and dismounting direct access volumes. Most of these functions are executed through SVC 1 and SVC 7 calls.

2.4.4 Console Support

The command processor controls the user task communication with the keyboard/printer device used as the system console. This is accomplished with a dummy driver which intercepts all log messages and SVC 1 requests to the console device and executes them for the user task. Because of this feature and the structure of task management, most commands can be

entered and executed while a user task is active, even if the task has assigned the console device.

2.5 FILE MANAGEMENT

The file management routines handle all access to bulk storage files, either by the user task or by the system. There are four basic modules in this package: the directory and bit-map handler, the contiguous file access method, the indexed file access method, and the SVC 7 processor. In addition, there is a set of utility programs.

2.5.1 SVC 7 Processor

This package processes all SVC 7 calls. It calls on the directory and bit-map handler when a file is assigned, allocated, deleted, or check-pointed. When a file is closed, it calls the disc driver, as required, to make sure all valid data is written on the disc. Protection keys are checked by this module. It also performs all assignment of devices to logical units.

2.5.2 Directory and Bit-Map Handler

This package handles all access to, and modifications of, the directory and bit map for each bulk storage device. Entries are provided to look up a file in the directory, to enter a new file name in the directory, to modify or delete a directory entry, to allocate one or more contiguous sectors of storage, or to release allocated bulk storage.

2.5.3 Contiguous File Access Method

This package is entered when an I/O request is made to a contiguous file. It performs sector address computations and enters the disc driver.

2.5.4 Indexed File Access Method

This package is entered when an I/O request is made to an indexed file. It handles all buffering and unbuffering, calls the disc driver for read or write whenever a buffer is filled or emptied, and allocates new space on the appropriate bulk storage device as required for file expansion.

2.5.5 Disc Utility Programs

Along with OS/16 MT2, the user is provided with a set of utility tasks which perform miscellaneous non-SVC 7 functions. This includes a disc compress, and a disc integrity check utility.

2.6 Floating Point Support

Support is provided for Single Precision Floating Point (SPFP) and Double Precision Floating Point (DPFP) as a system option. In addition optional traps are provided for the extended SPFP instructions available on a Model 8/16 with SPFP hardware, but on no other 16-bit processor.

In multi-tasking systems on processors with hardware DPFP, the DPFP registers are saved during scheduling. Software traps for DPFP instructions are provided for Models 7/16 and equivalent without DPFP hardware, the registers being maintained in the task's UDL.

SPFP support is functionally similar to DPFP support described above; the registers being saved or maintained in a separate area in the UDL.

The register save areas are reserved in the UDL as shown in Figure 2-5.

CHAPTER 3
SYSTEM CONVENTIONS

3.1 MACHINE STATES

OS/16 programs, tasks, and routines run in one of five defined states. These states are differentiated by a combination of PSW bits and flag bits of an active task. Any state not defined below is not permissible. At any given instant in time, the processor is executing in one of these states. They are, in increasing order of priority and privilege

1. User Task (UT)
2. Executive Task (ET)
3. Executive System Level (ESL)
4. Supervisor (SU)
5. Interrupt Service (IS)

The definition of these states in terms of PSW and TCB flag bits is shown in Table 3-1.

3.1.1 User Task State (UT)

The UT state is the state in which all user tasks run. The PSW protect bit is set. Internal and external interrupts are enabled, with the possible exception of the arithmetic fault interrupt bit, which is the only interrupt bit that is under user control. This state may only be exited through an interrupt or execution of an SVC.

3.1.2 Executive Task State (ET)

The ET state is the state in which all executive tasks (E-tasks) run (see Chapter 8). Protect mode is disabled. All interrupts, with the possible exception of arithmetic fault, are enabled. All SVCs are permitted. This state should only be exited through an interrupt or execution of an SVC.

3.1.3 Executive System Level State (ESL)

The ESL state is the state in which reentrant or roadblocked system code is executed on behalf of a task. Machine constraints are the same as for the ET state. The user's registers and PSW have been saved in a save area other than TCB.RSAV/TCB.CPSW.

All SVCs are permitted. This state may be exited in several ways:

- Return to UT/ET state by restoring the task's registers and current PSW and loading a PSW which goes to IOTERM in supervisor state.
- LPSW or EPSR that enters SU or IS state
- External or internal interrupt
- SVC

3.1.4 Supervisor State (SU)

The SU state is the state in which the system executes system code which is non-reentrant or is not roadblocked. Also executed in this state is code which changes critical system information such as the TCB. System queue service interrupts are disabled. No SVCs may be executed. This state is exited by a LPSW, EPSR, or external interrupt.

3.1.5 Interrupt Services State (IS)

The IS state is used for interrupt service routines within drivers and in the list instruction traps. All interrupts are disabled except machine malfunction and fixed point divide fault. This state is exited by a LPSW. Since all interrupts are disabled, it is extremely important that all IS code be as brief as possible.

TABLE 3-1 SYSTEM STATES

STATE	PSW Status Bits							TCB FLGS	TCB OPT
	EI 1	MM 2	DF 3	AS 4	FP 5	QS 6	PM 7	SL	ET
UT	1	1	d	1	d	1	1	0	0
ET	1	1	d	1	d	1	0	0	1
ESL	1	1	1	1	1	1	0	1	d
SU	1	1	1	1	1	0	0	d	d
IS	0	1	1	0	0	0	0	d	d

0 means bit must be zero
1 means bit must be one
d means bit may be zero or one
EI External Interrupt
MM Machine Malfunction
DF Divide Fault
AS Automatic I/O
FP Floating Point Fault
QS Queue Service
PM Protect Mode
SL System Level
ET Executive Task
UT User Task
ESL Executive System Level
SU Supervisor
IS Interrupt Service

3.2 SVC DEFINITIONS AND CONVENTIONS

SVC	Function	Type
1	I/O	I
2	General Service	II
3	End of Task	I
5	Fetch Overlay	II
6	Intertask Communication	II
7	File Management	II
9	TSW Swap	I

All SVC interrupts cause the system to enter the SU state. Type I SVCs enter directly to the appropriate service routines. Type II SVCs enter SVCEXEC which contains roadblocking logic. Type I SVCs execute in the SU state, thus eliminating the extra register save areas. Type II SVCs execute, for the most part, in the ESL state.

SVCEXEC passes control to an SVC executor with the:

1. Address of the SVC parameter block in register 2.
2. Address of the task control block of the invoking task in register 8.

It is the responsibility of the executor to perform validity checking of any address passed in the parameter block.

3.3 INTERNAL INTERRUPT CONVENTIONS

Internal interrupts cause control to be passed to EXEC in SU state. EXEC transfers control to the individual interrupt handler. In all cases, if the system includes the Command Processor module, a message is output to the system log indicating the nature of the interrupt and the address at which it occurred.

3.4 SUBROUTINE CONVENTIONS

Two levels of subroutine linkage are defined for system code. The mainline level is allowed to use the full set of registers (R0-RF). First level subroutines are linked through R8 and may use R8-RF without save/restore. Second level subroutines are linked through RC and may use RC-RF without save/restore. This is a general definition used as a guideline.

3.4.1 Calling Sequences

Parameters are passed in registers or in memory. Parameters may be passed in memory immediately following the BAL instruction. Parameters may be passed in system tables such as DCB, TCB, etc.

3.4.2 Exits

The normal exit from a subroutine should be either to the address contained in the link register, or to a specific number of halfwords past the address contained in the link register. Alternate exits must be to locations passed as parameters. Exits to unlabeled addresses are not permitted.

3.5 GENERAL NAMING CONVENTIONS

3.5.1 Data Structures

All data structures (defined by CAL STRUC statements) in OS/16 are named with three character symbolic names, e.g., TCB, SPT, DMT. All fields within these structures are defined by a name of the form SSS.FFF, where SSS is the strcuture name, and FFF is the field name. (See Appendix 2 for structure definitions).

3.5.2 Bits

Certain fields in a data structure contain flag bits to denote information. These flag bits are manipulated with logical immediate instructions (e.g., THI, OHI, NHI). For each flag bit there is a definition for the mask of the form SFFF.XXM: where S is a character which refers to the structure name, FFF are three characters which refer to the field, XX identifies the function of the flag bit, and M denotes a

bit mask. For example, in the TCB there is a field (TCB.OPT) which contains the option bits. Bit 0=1 means the task is an E-Task. The bit mask definition of this flag is:

```
TOPT.ETM EQU X'8000'
```

CHAPTER 4
SYSTEM OVERLAY SCHEME

4.1 INTRODUCTION

A disc-based OS/16 MT2 System may be overlayed internally to reduce the memory requirement for the system itself. There may be slight lessening of system response due to the greater number of disc transfers that occur.

There are four areas in the system that can be overlayed:

1. Command Processor: command executors
   error message output routine
   CSS routines
   system console processing routines

   This is requested through the CUP statement OVCMD. The SYSGEN parameter OVCMD is equated to 1 (overlay) to override the default 0 (do not overlay).

2. Intertask communication services within the executive:
   SVC 6 functions
   fetch overlay SVC 5

   This is requested through the CUP statement OVSIX. The SYSGEN parameter OVSIX is equated to 1 to override the default 0.

3. General supervisor services within the executive:
   SVC 2 functions except timer management

   This is requested through the CUP statement OVTWO. The system parameter OVTWO is equated to 1 to override the default 0.

4. File management services:

SVC 7 functions

This is requested through the CUP statement OVSEVEN. The SYSGEN parameter OVSEVEN is equated to 1 to override the default 0.

If the executive and/or file manager areas are overlayed, then the command processor is overlayed. This is set up by including the following statements in the source of each OS module:

```
OVERLAY EQU OVCMD!OVSEVEN!OVSIX!OVTWO
OVCMD   EQU OVERLAY
```

4.2 GENERAL SYSTEM CONSIDERATIONS

4.2.1 Size

The implemented overlay scheme does not increase the size of the non-overlayed system. The size of an overlay area is the size of the largest overlay to use that area, to the nearest halfword. This is true even if the DELETE SYSGEN option deletes part or all of what would have been the largest overlay.

4.2.2 Speed

An overlayed routine is not reloaded if it is already in memory when called. Features commonly used together are, size permitting, grouped together in the same overlay. An overlay is loaded with a standard SVC 1 read/wait request to the disc involving one disc transfer. For any given overlay area, the number of bytes read equals the number of bytes in the reserved area, i.e., in the largest overlay for that area.

4.2.3 Disc Space

The operating system, with all its overlays, is built on a contiguous disc file by TET/16. The overlays are built above the operating system in the order found i.e. in the order coded (see Figure 4-1). Each overlay for a given area begins on a sector boundary and occupies the smallest number of complete sectors that are required to hold the largest overlay for that area.

4.2.4 Flexibility

The overlay scheme allows for one or more routines per overlay, or one routine divided between two or more overlays. By reference to the sizes of the individual routines, the optimum arrangement is implemented.

4.2.5 The User Interface

For each area to be overlayed, the user enters one CUP statement at system generation. The operating system is built onto a contiguous disc file using TET/16. The file descriptor must be of the form:

```
SVOL:OS/16XXXX.NNN
```

where SVOL = volume name of OS disc pack

XXXX = up to four optional characters

NNN = three hexadecimal digits

The system is loaded with the OS/16 MT2 Bootloader (or LSU/ALO). The required version of the OS is specified by setting up the extension 0NNN in locations X'7E' to X'7F'. The bootloader searches for a file of the correct format, and if one is found, loads the OS image into memory and starts the system at address X'60'. The OS disc volume is automatically marked on-line with protect as if the operator had entered:

```
MARK   DISC:,ON,OS,PROTECT
```

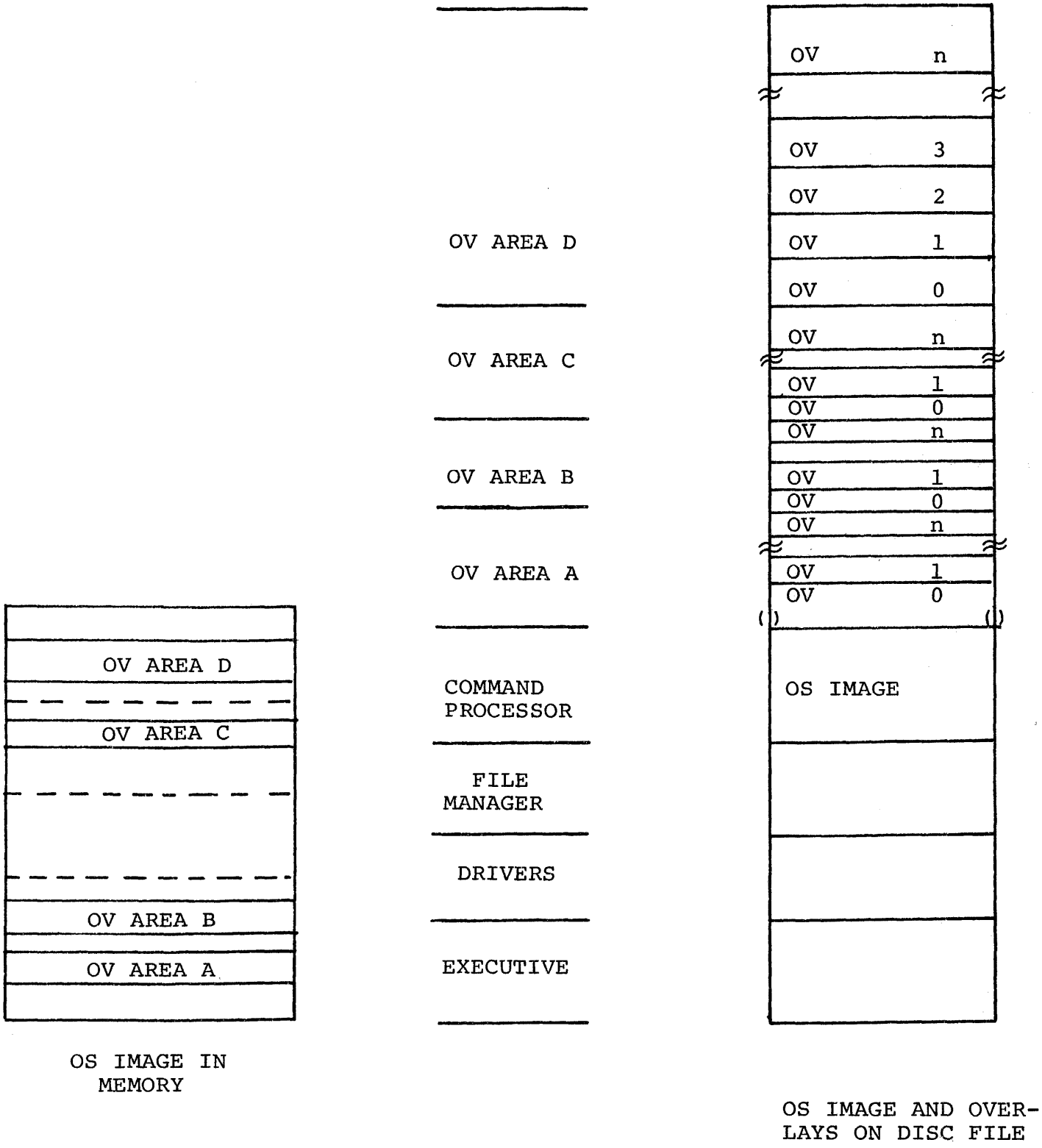


Figure 4-1 Organization of OS Image in Memory and on Disc

4.2.6 Roadblocking

Roadblocking prevents the operating system from attempting to service two concurrent SVC requests of the same type (eg. two SVC 2s). This is described in Chapter 5.

Roadblocking is particularly important in an overlayed system. During task scheduling, a lower priority task is interrupted if a higher priority task becomes ready. The lower priority task may have recently issued an SVC 2,5,6, or 7 request and caused an overlay to be loaded into memory. If no special action was taken, and rescheduling occurs while the overlayed SVC executor is operating on behalf of the task, it could be overwritten by a similar SVC request from the higher priority task. When control is returned to the first task at the point where it was interrupted, the correct overlay routine is no longer present.

Therefore, if a task requests an SVC service from SVC 2 functions, SVC 5 and 6 functions, or SVC 7 functions while another task in the system currently has access to that service, it is put into an SVC wait state until that service becomes available again.

4.2.7 Overlay Load Logical Unit:

Overlays are loaded by an SVC 1 bare disc read/wait request on the current task's file manager LU (number X'FF').

4.2.8 System Constants held in SPT:

SPT.FLBA	(OSOVST)	Random address of OS file on the disc (fullword)
SPT.OVFD	(OSOVFD)	File descriptor of OS file
SPT.SDCB	(OSOVDB)	DCB address for OS file

4.3 THE OVERLAY STRUCTURE

Routines to be overlayed can be positioned anywhere within a module, but they must be consecutive, i.e., there must be no resident code between the first and last overlays. Figure 4-2 shows how these are handled. If routines are not overlayed, they follow normally one after the other. If overlayed, an overlay handler is inserted at the beginning. Each overlay is positioned as from the start of the overlay area, and special information is inserted at the end to maintain control over the area at run-time.

No overlay can communicate directly with another overlay. It can, however, call the overlay handler to load an overlay over itself and pass control to it.

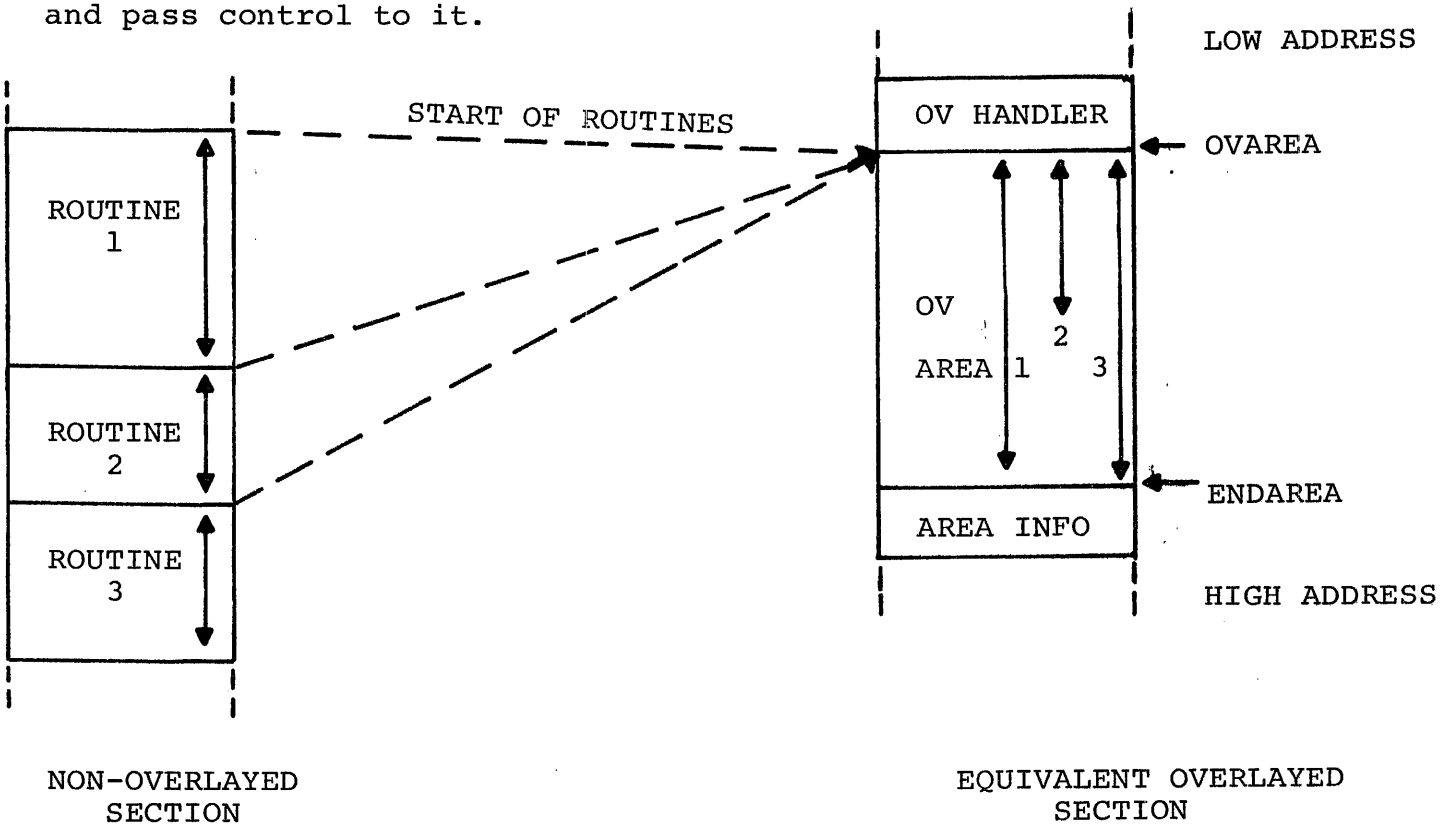


Figure 4-2 Comparison Between Non-Overlayed and Overlayed Routines

4.4 CODING CONVENTIONS

The non-overlayed OS contains only relocatable code, and is biased at zero when built by TET/16 or the library loader. An absolute 'ORG' statement takes on a special meaning when an operating system is built; it indicates the start or end of an overlay and is immediately followed by the true relocatable 'ORG' statement so that the standard object code processing can proceed once the overlay has been identified as such.

Considering one overlay area, let

OVAREA = start of overlay area

ENDAREA = end of overlay area

OVRPA = random address of start of first overlay in disc file

n = overlay number (first overlay = 0)

OVSIZE or m = number of sectors to hold largest overlay

CURROV = current overlay number (i.e., in memory)

OSOVST = random address of start of OS disc file (from start of disc).

The absolute 'ORG' statements are defined as follows:

```
ORG     0     start of first/next overlay
ORG     2     end of all overlays for this area
```

At the beginning of the area to be overlayed, OVAREA and ENDAREA are initialized as follows:

```
OVAREA  EQU # *
ENDAREA EQU # *
```

As each overlay is assembled ENDAREA is maintained at the end of the largest overlay so far:

```
          ORG     0
          ORG     OVAREA
          ((code for one overlay))
          IFP     *-ENDAREA
ENDAREA   EQU     *
          ENDC
```

At the end of the area to be overlayed, the following resident code must be positioned from the end of the largest overlay and not necessarily from the end of the last overlay. This code completes the area:

```
          ORG     2
          ORG     ENDAREA
OVRPA     DS      2
CURROV    DC      -1
OVSIZE    EQU     ENDAREA-OVAREA+255/256
```

See Figure 4-3 for an example of the conditional code inserted into the command processor. Each overlay area has its own set of unique symbols. The following shows how the example given corresponds to the general case described above:

GENERAL CASE SYMBOLS	COMMAND PROCESSOR EXAMPLE
OVAREA	OVCMD
ENDAREA	ENDOVC
OVRPA	OVCRAD
CURROV	CURRC
OVSIZE	OVSIZE

```
          IFP       OVCMD
OVC       EQU       *
ENDOVC    EQU       *
*
* OVERLAY 0
*
          ORG       0
          ORG       OVC
          ENDC

     ((Routines for first overlay))

          IFP       OVCMD
          IFP       *-ENDOVC
ENDOVC    EQU       *
          ENDC
*
* OVERLAY 1
*
          ORG       0
          ORG       OVC
          ENDC

     ((Routines for next overlay))

.    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .
.    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .

          IFP       OVCMD
          IFP       *-ENDOVC
ENDOVC    EQU       *
          ENDC
*
* OVERLAY 15
*
          ORG       0
          ORG       OVC
          ENDC

     ((Routines for last overlay))

          IFP       OVCMD
          IFP       *-ENDOVC
ENDOVC    EQU       *
          ENDC

*
* NO MORE OVERLAYS TO USE THIS AREA
          ORG       2
          ORG       ENDOVC
*
* SPECIAL INFORMATION ABOUT AREA - RAD OF START OVS ON DISC
*                                - EQUATE OF NO OF SECTS TO HOLD 1 OV
*
OVCRAD    DS        2
CURRC     DC        -1
OVSIZE    EQU       ENDOVC-OVC+255/256
          ENDC
```

Figure 4-3 Example of Conditional Overlay Code

4.5 THE OVERLAY HANDLER

The overlay handler requires the following information to load an overlay:

1. The number of the overlay required (n). This is indicated by the calling routine.
2. The number of the overlay currently in memory (CURROV). The current overlay number is preset -1 and updated by the overlay handler as different overlays are brought in.
3. The random address of the disc from which to load the overlay. This calculated as

   OSOVST+OVRPA+(n*m)

   where m = ENDAREA-OVAREA+255/256
   OSOVST = set up by Bootloader when loading OS.
   OVRPA = set up by TET/16 when building OS.
4. The number of bytes to load. The SVC 1 read/wait parameter block loads from (OVAREA) to (ENDAREA-1) inclusive.

4.5.1 Entry Interface

There are two entry points.

1. For a routine called off a multi-switch.

   Example: Command executor

   Register x: Index, formed from multi-switch, to point into table of (n*m) values for each overlay.

   Register y: Address to branch to once the overlay is in memory.
2. For a known routine - no multi-switch involved.

   Example: Linking a routine divided into two or more overlays.

   Register x: The value (n*m) for the overlay

   Register y: Address to branch to once the overlay is in memory.

'x' and 'y' for the current OS depend upon the particular overlay handler (one per area).

4.5.2 General Outline of Flow through Handler.

From first entry point:

- Form index into byte table and access (n*m).

From second entry point:

- If this overlay is currently in memory, then exit.
- Save essential registers.
- Form random address of overlay on disc relative to start of file.
- Call common subroutine OVLU to set up LU and calculate random address of overlay relative to start of disc.
- Issue SVC 1 read/wait request (bare disc I/O).
- Check status. If error, exit to error handler OVIO.
- Restore registers.
- Branch as directed on entry.

4.5.3 Set up Overlay Load LU and Random Address (Routine OVLU).

This routine picks up the OS file DCB address from the SPT (OSOVDB) and checks it. If it has not been set up (zero), it exits immediately with CC = 0. Otherwise, it stores the DCB address in the file manager LU slot in the current task's TCB (TCB.FMLU) and modifies the device attributes so that read random is allowed. It updates the random address given (relative to start of file) to the address, relative to the start of the disc by adding the fullword value OSOVST from the SPT. It then exits with CC $\neq$ 0.

4.5.4 Process Overlay Load I/O Error (Routine OVIO).

This routine displays X'99999999' pattern on display panel and exits to retry the I/O. Both OVLU and OVIO are in the executive module.

4.5.5 Example of Overlayed Module

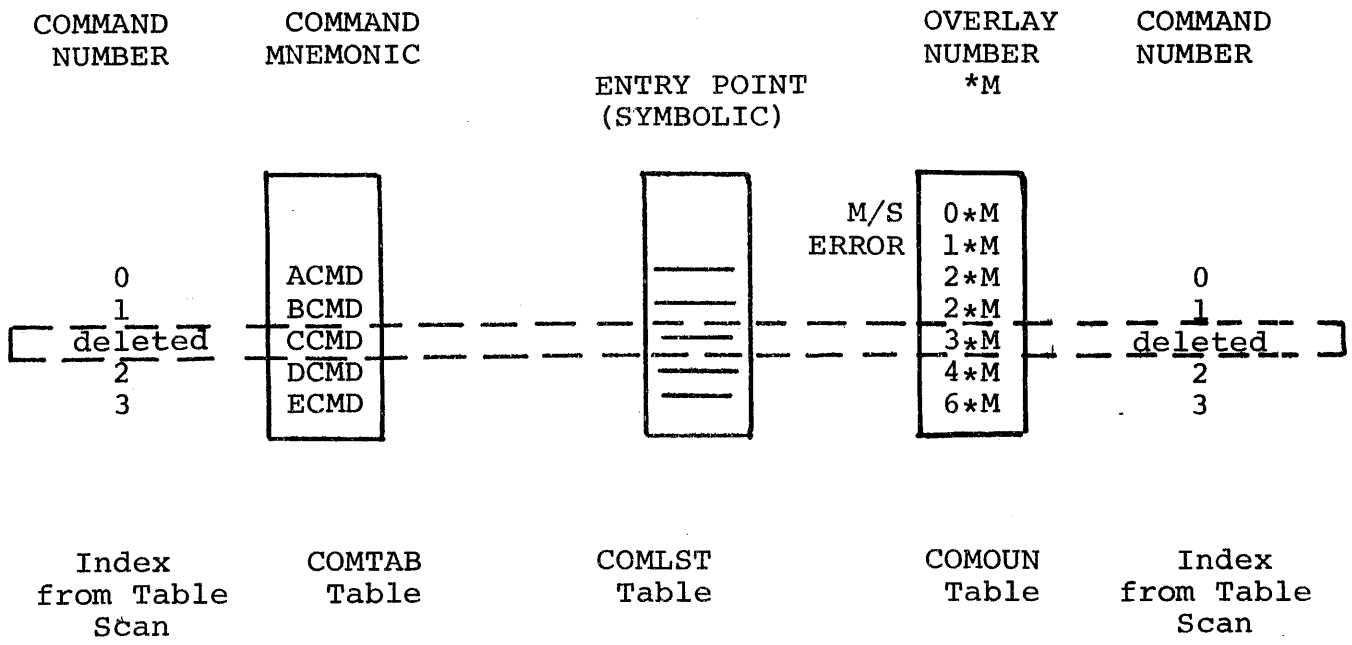
Figure 4-4 illustrates the various tables associated with a very simple command processor. In this example there are five operator commands. These are listed in the standard form of a mnemonic table. In parallel with this table is a table of symbolic entry points for the executors and a table of (n*m) constants.

Other routines to be overlayed (such as an error message output routine) are not called from the multi-switch. These are represented in the (n*m) table to be accessed directly, if required.

The routines are not evenly distributed among the overlays; one overlay holds two routines, another only part of one routine.

Note that one or more commands can be deleted as long as the corresponding items in all three tables are removed, and the conditional delete statements around the executor itself are placed within the ORG statements defining the overlay, i.e., the overlay exists by number, but is of zero size.

If one routine of a number called off a multi-switch must be resident, even though the rest are overlayed (example: timer management SVC 2 code 23), the routine is placed beyond the end of the overlay area and is identified by a negative value (-1) in the byte table of (n*m) values.



1st overlay - command identification and multiswitch
2nd overlay - error message O/P
3rd overlay - routines A and B
4th overlay - routine C (deleted in example)
5th overlay - first part routine D
6th overlay - second part routine D
7th overlay - routine E

Figure 4-4 Tables Associated With a Very Simple Command Processor

4.6 THE ROUTINES TO BE OVERLAYED

Routines overlayed into the same area, but into different overlays, cannot communicate with each other as only one of them can be in memory at any one time. Only certain areas of the system, in practice, can be overlayed. Individual command executors and SVC handlers may be overlayed without imposing too great a load on the system.

Routines designated to share one area are grouped together or divided up to form a number of overlays. Small routines commonly used together are grouped together. If one routine is far larger than all the others

(example: SVC 6 image loader), then it may be worth dividing it up into two or three overlays to reduce the size of the overlay area even though this involves two or three disc transfers. Here, memory savings is weighed against the overhead. Routines which loop around from the end to the beginning to repeat the same process a number of times are not divided up no matter how large (example: the FILES output loop).

'ORG' statements must not be used within overlay routines. Any absolute 'ORG', or relative 'ORG' which takes the current location counter out of the overlay area corrupts the system.

Any EXTRN symbol referenced in an overlay must be of the form XXXXNN

where XXXX = any four alphanumeric characters following standard conventions

NN = two digit decimal number of the overlay itself.

During assembly, CAL chains together all external references to a given EXTRN symbol giving the end of the chain as one of the last items in the object code (see Figure 4-5). If the chain is allowed to pass through overlayed code, TET/16 is passed an address known to be within the overlay area, but with no indication of which overlay. Therefore, there must be a unique chain for each overlay and one for the resident code, i.e., unique symbols must be used within each segment. The last two characters of the EXTRN symbol must be the overlay number (decimal).

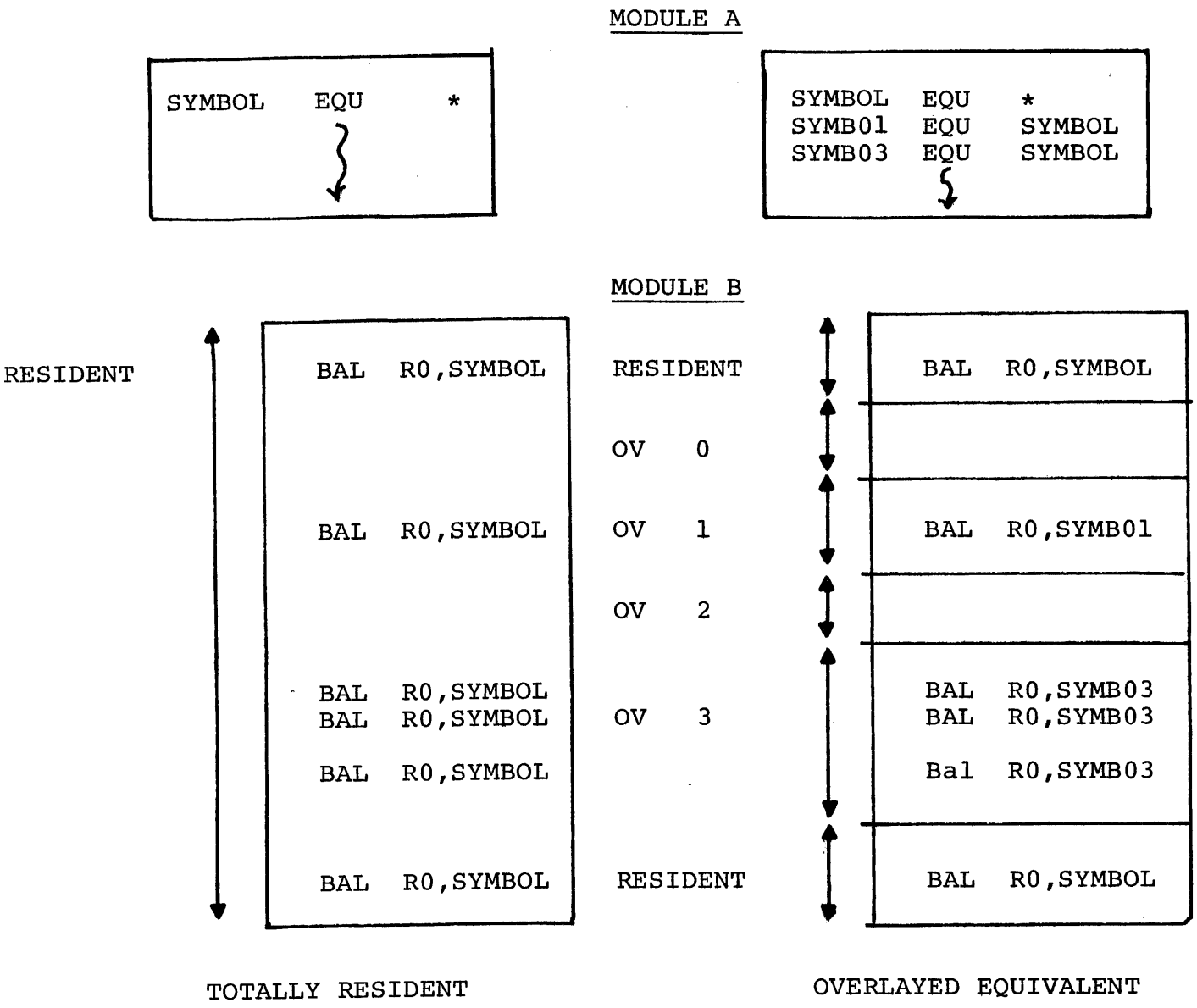


Figure 4-5 Example of use of EXTRNs in Overlays

More information may be gained from the source listings themselves. As an aid to locating particular subroutines or data items associated with the overlay structure, Figure 4-6 lists symbols of interest used in each of the four possible overlay areas:

- command executors in the command processor
- SVC 2 executors in the executive
- SVC 5 and 6 executors in the executive
- SVC 7 executors in the file manager

	COMMAND PROCESSOR	EXECUTIVE		FILE MANAGER
	EXECUTORS	SVC 2	SVC 5,6	SVC 7
Overlay Handler	LDC.T LDC.N	LDS2.T LDS2.N	LDS6.T LDS6.N	LDS7.T LDS7.N
Routines used	OVLU OVIO	OVLU OVIO	OVLU OVIO	OVLU OVIO
SVC 1 parameter block	SVCOVC	SVCOV2	SVCOV6	SVCOV7
Mnemonic table	COMTAB	___	___	___
Entry point table	COMLST	STAB	SVC6.FTB	S7.CTB
Byte table (n*m)	COMOVN	S2OVN	S6OVN	S7OVN
Start of area	OVC	OV2	OV6	OV7
End of area	ENDOVC	ENDOV2	ENDOV6	ENDOV7
Random Address of first overlay	OVCRAD	OV2RAD	OV6RAD	OV7RAD
Current overlay	CURRC	CURR2	CURR6	CURR7
Overlay size	OVSIZE	OV2SIZ	OV6SIZ	OV7SIZ
Overlay flag	OVCMD	OVTWO	OVSIX	OVSEVEN

Figure 4-6 Overlay Area Symbols

CHAPTER 5
EXECUTIVE DESCRIPTION

5.1 TASK MANAGEMENT

5.1.1 Task Control

In OS/16 MT2 a task may be in wait state or in ready state. The wait state indicates that some occurrence must take place before the task may proceed. The ready state indicates that all such necessary occurrences have taken place. Tasks are controlled through the use of several control blocks (see Figure 5-1).

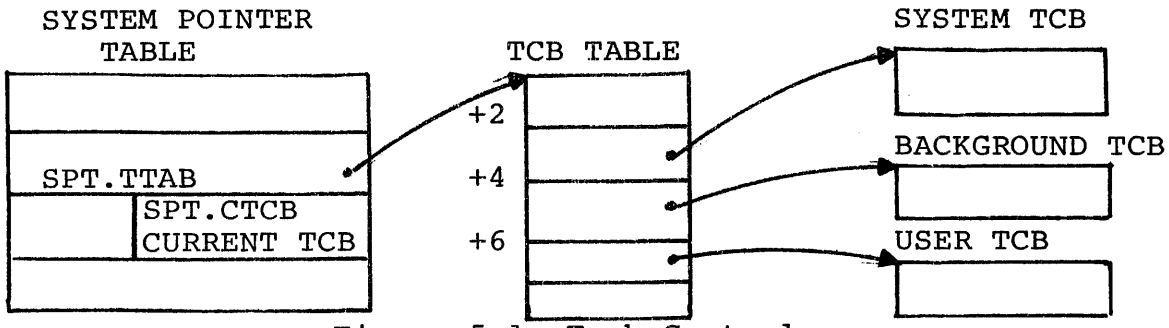


Figure 5-1. Task Control

Each task is described by a Task Control Block (TCB). The addresses of the TCBs are maintained in the TCB table which is pointed to by the System Pointer Table (SPT). A table is maintained consisting of the TCB indexes of all TCBs in priority order. (See Figure 5-2.) In OS/16, the system task has priority zero (highest), and user tasks may be assigned a priority between 10 and 249, inclusive. TCBs are referenced by their address, by their index into the TCB table, or by their number. The TCB number is equal to the TCB index divided by two. The system task is TCB number 1.

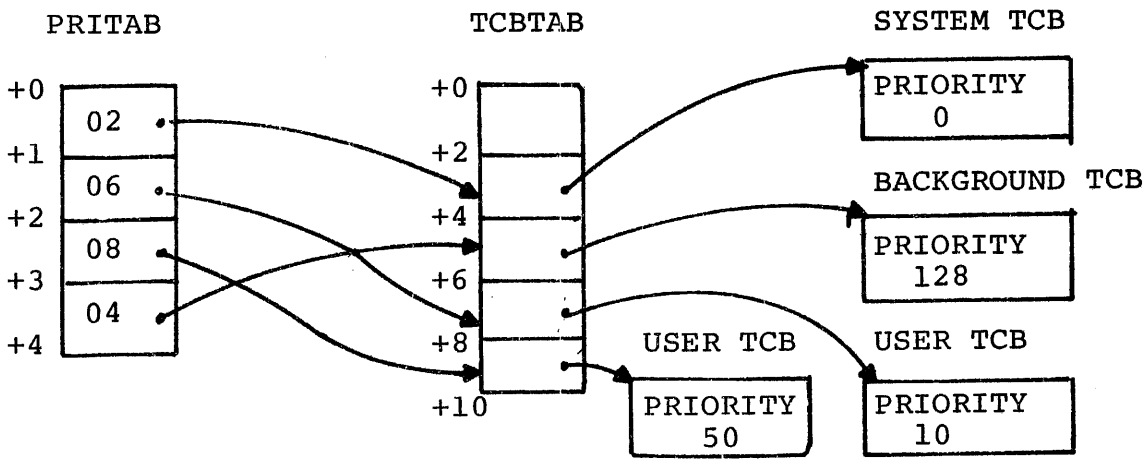


Figure 5-2 Priority Table

5.1.2 Task Management Routines

Task management routines are provided to schedule the highest priority ready task, arrange the priority table in priority order, set up memory protect patterns in all TCBs, and to cancel a task. These routines are detailed in the following paragraphs.

5.1.2.1 Schedule The Highest Priority Ready Task (SCHED) - If round-robin scheduling is SYSGENed, SCHED places the last ready task at the bottom of its priority in the priority table (PRITAB). This action causes task scheduling to be rotated between tasks of the same priority.

The priority table (PRITAB) is scanned and the first ready task (TCB.STAT=0) is activated. The active task pointer (ACTTCB or SPT.CTCB) is set to the TCB index, the task's general and floating point registers are restored from TCB.RSAV and UDL.FPRS respectively, and the task current PSW (TCB.CPSW) is loaded. If no ready task is found, ACTTCB is zeroed and a wait PSW is loaded with all interrupts enabled.

5.1.2.2 Arrange The Priority Table In Priority Order (PLINK) - PLINK uses a shuttle sort to sort the priority table (PRITAB) in priority order.

5.1.2.3 Set Up Memory Protect Pattern (MEMPT) - MEMPT sets up the memory protect pattern (TCB.MPT) in all TCBs. The pattern consists of four halfwords. Each bit represents 1 KB of memory. A bit set (1) indicates that the corresponding KB of memory is protected. A bit reset (0) indicates that the corresponding KB of memory is unprotected. If a task attempts a write into a protected area of memory, the Memory Protect Controller (MPC) interrupts the processor. The memory protect pattern for the background task is set up specifying all of memory protected except the background area. For all foreground tasks, the pattern is set up specifying all of memory protected except the foreground area.

5.1.2.4 Cancel A Task (CANCEL) - CANCEL sets the cancel pending flag (TFLG.CPM). If the task is in I/O wait (TSTT.IOM), it is removed from the I/O wait thread. All ongoing I/O is cancelled by an SINT into the driver's abort routine (DCB.ALOC). Console I/O is cancelled by zeroing CIORQ and CMDLMW entries that were created by this task. The UT/ET level PSW is set pointing to an SVC 3,255 instruction (ET.3) and the task is set ready. The next time the task is scheduled, the SVC 3,255 is executed and the SVC 3 handler is entered.

5.1.3 Task Status

The status of a task is defined by the setting of the bits in the status field of the TCB (TCB.STAT) and the value of the current TCB field of the SPT (ACTTCB or SPT.CTCB). Table 5-1 indicates detailed states and their meanings.

TABLE 5-1 TASK STATES

State	Indication	Meaning
Dormant	Dormant bit (TSTT.DMM) set in TCB.STAT	Task has not been started or has gone to end of task (EOT).
Ready	TCB.STAT=zero	Task will be dispatched when it becomes the highest priority task in this state.
Current	TCB index in SPT current TCB field	Task is the executing task.
Executive System Level (ESL)	System level (TFLG.SLM) set in TCB.FLGS	Task's registers and PSW saved in alternate save area. Task may be current, ready, or in wait.
Wait	A wait bit is set in TCB.STAT	Task needs a specific occurrence to take place before it may proceed.

5.2 SVC HANDLER

5.2.1 SVC1 Executor (SVC1)

Entry to SVC1 is in the SU state. The caller's general and floating point registers are saved in TCB.TSAV and UDL.FPRS, respectively. The updated PSW, with the condition code set to zero, is saved in TCB.CPSW. The Logical Unit (LU) specified is checked, and if valid, the address of the DCB/FCB is loaded from the slot in TCB.LTAB corresponding to the LU number. If the slot contains zero, the LU is not assigned and error status

is returned. If the DCB address is odd, the request is directed towards either the console reader, a teletype or carousel reader, or the second cassette of a cassette pair. The function code is validated using the attributes of the assignment in the TCB (for data transfer requests) or the attributes field of the DCB/FCB (for command function requests).

If the request specifies halt I/O to a device that supports halt I/O, and is busy with I/O on the specified LU, the I/O is aborted by an SINT into the driver's abort routine (DCB.ALOC). The parameter block status is set to zero. Exit is to IOTERM.

For FCB requests, the FCB in use flag (FFLG.IUM) is checked. If set, the function code is tested. If the command bit, the read bit, the write bit, or the wait bit is set, BUSY1 is entered. If test I/O complete is specified, the callers condition code is set to X'F' and SVC1 exits to IOTERM. If the FCB in use flag is reset, and wait only or test I/O complete is specified, the status is set to zero and exit is made to IOTERM. For data transfer requests, the start and end addresses are validated with two calls to ADDCHK1. The FCB in use flag is set and the proper disc driver is entered.

For DCB requests, the DCB busy flag (pointed to by DCB.BUSY) is checked. If set, BUSY is entered. If reset, and wait only or test I/O complete is specified, the status is set to zero and exit is made to IOTERM. For data transfer requests, the start and end addresses are validated.

For DCB requests, the DCB busy flag (pointed to by DCB.BUSY) is checked. If set, BUSY is entered. If reset, and wait only or test I/O complete is specified, the status is set to zero and exit is made to IOTERM. For data transfer requests, the start and end addresses are validated.

The DCB busy flag is set and the driver initialization routine (DIR) is entered with the following registers set up:

R1	Address of DCB
R2	TCB index in upper byte
R3	Address of parameter block
R4	Function code/logical unit
R6	Device number
R7	Zero (status)
R8	Address of DIR
R9	Address of busy flag

5.2.1.1 Wait Thread - The second byte of the first halfword in DCB.RSAV (lower byte of R2) is the wait thread pointer. This is a pointer to the first task's TCB in I/O wait for the device represented by the DCB. This TCB contains a pointer to the next TCB waiting for the device. The last TCB in the thread has a zero as its pointer. The pointer is called TCB.WTHR. Each TCB in the thread has the address of the beginning of the thread (DCB.RSAV+1) in TCB.WTAD.

When a driver is entered, the lower byte of R2 contains zero. This is stored into DCB.RSAV when the driver saves it's registers.

5.2.1.2 Device Busy - If a device is busy on an SVC 1 request, BUSY is entered. The function code is tested. If it is a command request,

BUSY1 is entered. If it is a wait only or a test I/O complete, the task that made the device busy are compared against the task and logical unit on which the wait only or test I/O is requested. If it is the same task and logical unit, BUSY7 is entered.

BUSY1 is entered if a device is busy on a read or write request or if an FCB is in use on a file request. The unconditional proceed (test I/O complete) bit is checked in the function code. If reset, BUSY7 is entered. If set, the callers condition code is set to X'F', the Wait bit is set in the function code register so no I/O proceed trap is generated, and exit is made to SETSTA.

BUSY7 is entered if a device is busy on a command request or a non-unconditional proceed read or write request. It is also entered from the disc driver if the stop disc I/O flag (STPDIO) is set. See the section on release FCB in Chapter 7 for the use of this flag. The task's current PSW is backed up to point to the SVC instruction. If the device is not a disc, IOWAIT is entered. If it is a disc, the stop disc I/O flag is checked. If reset, IOWAIT is entered. If set, the task is put into SVC 7 wait instead of I/O wait so that the I/O is re-issued after the release FCB has been completed. BUSY7 then exits to IOTERM.

5.2.1.2 Test Wait - When driver initialization is completed, IOTWAT is entered. The function code is checked. If I/O and wait is specified IOWAIT is entered, otherwise, IOTERM is entered.

5.2.1.4 Set Wait - IOWAIT is entered when a task is to be placed on the wait thread and in I/O wait. A pointer to the wait thread is placed in

TCB.WTAD. The task is added to the end of the thread by putting its task pointer in place of the zero, and setting its TCB.WTHR to zero, indicating the new end of the thread.

5.2.1.5 Remove Wait - When driver termination is completed, IODONE is entered. All tasks in the wait thread for this device are taken out of I/O wait. If the request was to an FCB, the FCB in use flag (FFLG.IUM) is reset. If the request was to a disc, and the stop disc I/O flag (STPDIO) is set, the number of outstanding disc I/Os (NUMDIO) is decremented. If it is decremented to zero, the file manager is taken out of SVC 7 wait. This allows the release FCB to complete. Exit is made to SETSTA.

5.2.1.6 Set Status - SETSTA is entered to set the SVC 1 status and length of data transfer fields. The device independent and device dependent status registers (R7 and R6) are stored in SVC1.STA in the parameter block. The length of data transfer is returned to non-compatable tasks on read or write requests. If the request did not specify wait, an I/O proceed termination trap is generated by a call to ADTSKQ. Exit is made to IOTERM.

5.2.1.7 System Queue - The system queue is a circular list in standard INTERDATA format. The number of slots is determined by the number of devices in the system. The system queue is used, along with the system queue service interrupt, to establish a non-reentrant supervisor state (SU state) in which no task may be scheduled, but device interrupts can still be serviced. When a driver completes its transfer, it adds its DCB address (plus one) to the system queue and exits to DCB.LEAV. If queue service interrupts were enabled at the

time the last ISR was entered, the system queue is serviced immediately and the driver termination routine is entered. When the termination routine is complete, the task scheduler is entered from IODONE to schedule the highest priority ready task. If queue service interrupts were disabled, the system queue is not serviced until the code executing in the SU state changes to the ET or ESL state. In this way, code executing in the SU state is always assured that it is never re-entered because no task can execute instructions that may cause it to be entered.

IOTERM is entered when internal interrupt handlers and supervisor call handlers terminate. It checks the system queue (LIOTRM) in case any items were added while the system was in the SU state. If there is nothing on the queue, the scheduler (SCHED) is branched to. If an item is removed from the queue, it is checked for zero. If zero, IOTERM is branched to in order to try again. A zero item is added if an ESL state routine wants to put itself into wait, as just setting a bit in TCB.STAT does not stop the routine until an interrupt occurs. The zero item on the system queue causes a queue service interrupt to occur and the scheduler to be entered.

If the queue item is not zero, it is checked to see if it is odd or even. If the queue item is odd, it is a standard I/O driver termination and the registers are picked up from DCB.RSAV. The termination routine is branched to whose address is in DCB.TERM. If the queue item is even, it is a clock termination or a system console termination. The DCB address is picked up from the queue item plus eight, and the termination routine is entered.

5.2.1.8 Seek Queue - The seek queue (SEEKQ) is used by the moving head disc driver to regain the SELCH after a seek. While the disc head is seeking, the SELCH is not needed, so its busy flag is reset.

When the seek completes, the SELCH is again needed so the busy flag is checked. If the SELCH is free, the busy flag is set and the transfer starts. If the SELCH is in use, the disc device number is added to the seek queue and the driver exits to DCB.LEAV.

When an I/O terminates (IODONE), and when a device is busy (BUSY), the seek queue is checked. If there is an item on the queue, the driver is entered by an SINT on the device number. The transfer then proceeds as normal.

5.2.2 SVC Executive

The SVC executive, SVCEXEC, is entered when a task issues an SVC 2, 5, 6, or 7. It handles roadblocking of SVC executors. The SVC executors are non-reentrant, but are executed in a reentrant state (ESL state). Roadblocking is used to keep other tasks from entering an SVC executor that is in use.

SVCEXEC checks the roadblock table item (RDBLK) corresponding to the SVC number. If the SVC is in use (RDBLK entry non-zero), the calling task is put into SVC n wait, where n is the requested SVC. Exit is then made to IOTERM. If the requested SVC is free, the TCB index of the calling task is put into the roadblock table. The task's general registers, current priority, and current PSW are saved in the save area corresponding to the SVC requested. The priority in the task's TCB is set to one. The SVC executor is then entered in the ESL state.

When a roadblocked SVC executor completes, either normally or because of an error, the SVC return routine (SVCnRTN, where n is the SVC number is entered. It takes all tasks out of SVC n wait, where n is the SVC number. The roadblock table entry is set to zero. The task's general registers and current PSW are restored to the TCB. If a task trap was to occur while the task was in the ESL state, the TSWs are swapped. The task's current priority is restored to TCB.CPRI. Exit is made to IOTERM.

5.2.3 SVC 2 Executor (SVC2)

Entry to SVC2 is in the ESL state from SVCEXEC. The parameter block address is checked to ensure that it is within the partition. The function code is validated, the proper overlay is brought in, and the executor is branched to.

5.2.3.1 SVC 2 Overlays - The SVC 2 routines are overlayed in nine parts. SVC 2 code 1 and SVC 2 code 23 are not overlayed. The SVC 2 overlay loader (LDS2) loads the required overlays.

5.2.3.2 SVC 2 Overlay Table - Table 5-2 shows the SVC 2 executors in each overlay.

TABLE 5-2 SVC 2 OVERLAYS

OVERLAY	CODE	NAME	DESCRIPTION
0	0	S2JRNL	Journal Call
	2	S2GETS	Get Storage
1	3	S2RELS	Release Storage
	4	S2SETS	Set Status
	5	S2FPTR	Fetch Pointer
	20	S2EXP	Expand
	21	S2CON	Contract
2	6	S2UPAK	Unpack
3	7	SVCLG	Log Message
	8	SVCLK	Fetch Time
	9	SVCDT	Fetch Date

TABLE 5-2 SVC 2 OVERLAYS (Continued)

OVERLAY	CODE	NAME	DESCRIPTION
4	15	S2PAK	Pack Numeric
5	16	S2PKFILE	Pack File Descriptor, part 1
6	16	PKFL2	Pack File Descriptor, part 2
7	17	S2SCAN	Scan Mnemonic Table
8	18	S2MOV	Move Characters
	19	S2PEEK	Peek
Not Overlayed	23	S2TIME	Timer Management
	1	S2PAUS	Pause

5.2.4 SVC 3 Executor (SVC3)

Entry to SVC3 is in the SU state. The cancel pending flag (TFLG.CPM) is set in TCB.FLGS. The effective address of the SVC is stored as the return code in TCB.RC. If the task is a non-resident foreground task, the command processor current task is cleared if it is set to this task. The current TSW is set to zero. If the task has a time interval outstanding, it is cancelled. SVC3 then enters the ET state. All reads are cancelled with an SINT into the driver's abort routine (DCB.ALOC). SVC 7 checkpoint calls for resident tasks, or SVC 7 close calls for non-resident tasks, are issued on each assigned LU. This ensures that all writes have been completed, that the abort of all reads is complete, and that all files are in a checkpointed condition. If the task is currently connected to a trap-generating device, the connection is frozen and unconnected by a call to the TGD driver. The return code is unpacked with an SVC 2. SVC3 then returns to the SU state. The task is set dormant. Cancel pending is reset in TCB.FLGS. ATEXT is called to output the end of task message. If the task is resident, SVC3 exits to IOTERM. Otherwise, TCB.ID, TCB.OPT and TCB.FLGS are

cleared. TCB.CPRI is set to 255 to speed up task scheduling. If the task was to be rolled out, the roll is no longer required so the task that caused the roll attempt is taken out of SVC 6 wait. If a roll in is required (PARTAB contains an odd address), the address of CROLL is added to the command processor queue by a call to ADTSKQ. SVC3 exits to IOTERM when complete.

5.2.5 SVC 5/6 Executor (SVC6)

The SVC 5/6 executor, SVC6, is entered in the ESL state from SVCEXEC. It decides which SVC call it is, loads the proper overlay, and branches to it.

5.2.5.1 SVC 5/6 Overlays - The SVC 5/6 routines are overlayed in seven parts. See Table 5-3 for the SVC 5/6 executors in each overlay. The SVC 6 overlay loader (LDS6) loads the required overlays.

TABLE 5-3 SVC 5/6 OVERLAYS

OVERLAY	ROUTINES	DESCRIPTION
0	SVC5R	SVC 5 Handler
1	SVC6R	SVC 6 Initialize and Executor Table
2	SVC6.ET	SVC 6 End Task
	SVC6.FIX	SVC 6 Fix
	SVC6.ATQ	SVC 6 Add to Task Queue
	SVC6.SEN	SVC 6 Send message
	SVC6.PRR	SVC 6 Priority
	SVC6.CON	SVC 6 Connect
3	SVC6.TGD	SVC 6 Thaw, Sint, Freeze, Unconnect
	SVC6.UNF	SVC 6 Unfix
	SVC6.STR	SVC 6 Start
4	SVC6.LT	SVC 6 Load, part 1 - read LIB
5	LT.CS2	SVC 6 Load, part 2 - rollout
6	LT.PA	SVC 6 Load, part 3 - load task

5.2.5.2 SVC 5 Handler (SVC5R) - The SVC 5 handler is called to load a user overlay. It is entered in the ESL state. The LU specified in the parameter block is placed in an SVC 1 parameter block (S5.S1). The option field is checked. If load after rewinding is specified, an SVC 1 is issued to rewind the LU. The overlay's LIB is read into the TCB register save area, TCB.RSAV, effectively reading it into the registers. The overlay name in the LIB is verified with the name in the SVC 5 parameter block. A test is made to determine if available storage for the overlay exists. LOAD.MOD is then called to load the overlay. After the overlay is loaded, CHKSUM is called to validate the task's checksum. Exit is made to SVC6RTN.

5.2.5.3 SVC 6 Initialize (SVC6R) - SVC6R is entered in the ESL state. If the SVC 6 was from the background task, the SVC 6 continue bit (TOPT.S6M) is checked in TCB.OPT. If set, the parameter block status is set to zero and SVC6RTN is entered to effectively ignore the call. If the option bit is reset, SVC6IL is entered to indicate an illegal SVC.

The direction field in the function code is checked. If the call is directed towards another task, the TCB is located by a call to LOCT. If the function specifies load, the directed task must not exit unless it is the background task. For all other functions, the task must exist.

After the directed task's TCB is located, the current status and priority are moved to SVC6.TST and SVC6.RPI respectively. SVC6.IF is then entered.

5.2.5.4 SVC 6 Function Determination (SVC6.IF) - SVC6.IF tests each bit of the function code, loads and branches to the executors requested. The executors are entered in the SU state with the following registers set up:

R0	Current position pointer into function code
R1	TCB address of calling task
R2	Address of SVC 6 parameter block
R3	TCB address of directed task
R4	Pointer into function code table
R5	Increment (2)
R6	End of function code table
R7	Current halfword of function code
R14	DCB address of Trap Generating Device (TGD)

The executors may use R8 through R15 as work registers. Executors return to SVC6.IF by issuing a LPSW of SVC6.RT. Error exit is made by placing the error status in R15 and branching to SVC6.ERR.

5.2.5.5 SVC 6 Function Executors

- END TASK (SVC6.ET) - LOCCHK is called to verify that the directed task is still present. The function code is checked for a delete task. If set, the task is made non-resident. CANCEL is called to cancel the task. If the call is self directed, SVC 6 processing is terminated by exiting to SVC6RTN.

- FIX (SVC6.FIX) - LOCCHK is called to verify that the directed task is still present. The memory resident bit (TOPT.MRM) is set in TCB.OPT.

- ADD TO TASK QUEUE (SVC6.ATQ) - LOCCHK is called to verify that the directed task is still present. ADTSKQ is called to add the parameter. Error code 12(C) is returned if any problems occur.

- SEND MESSAGE (SVC6.SEN) - LOCCHK is called to verify that the directed task is still present. ADD.BC is called to validate the message ring (UDL.MSGR) and message buffer (SVC6.MSG) addresses. If the current buffer in the message ring is empty (bit 15=0), the caller's task ID followed by a 64-byte message are moved into the ring buffer. The buffer is marked full (bit 15=1) and the address of the next buffer is placed in UDL.MSGR.

  ADTSKQ is called to queue the send message buffer address. If ADTSKQ returns CC=0, normal exit is made. Otherwise, the buffer address is restored in UDL.MSGR and the buffer is marked empty (bit 15=0). Any error results in a status of 11(B).

- SET PRIORITY (SVC6.PRR) - LOCCHK is called to verify that the directed task is still present. For a U-task, the desired priority is checked to ensure it is greater than 250. The priority is checked with the maximum priority. The UT/ET priority is updated and PLINK is called to adjust the priority table (PRITAB). Error code 5 is returned if the specified priority is invalid.

- CONNECT (SVC6.CON) - LOCCHK is called to verify that the directed task is still present. If the device is not connected

the TCB pointer and parameter are placed in the DCB. Error code 15(F) is returned if the device is already connected.

- THAW, SINT, FREEZE, UNCONNECT (SVC6.TGD) - LOCCHK is called to verify that the directed task is still present. A check is made to verify that the directed task is connected to the specified device. The TGD driver is entered to perform the requested function. For an unconnect, the task pointer in DCB.RSAV is set to zero. Error code 16(10) is returned if the device is not connected to the directed task.

- UNFIX (SVC6.UNF) - LOCCHK is called to verify that the directed task is still present. The memory resident bit (TOPT.MRM) is reset in TCB.OPT.

- START (SVC6.STR) - LOCCHK1 is called to verify that the directed task is still present. Error code 1 is returned if the call is self-directed. Error code 7 is returned if the directed task is not dormant or paused. The initial TSW (TCB.ITSW) is moved to the current TSW (TCB.CTSW). The current PSW (TCB.CPSW) is set up to ETSTAT or UTSTAT depending on OPTION ET (TOPT.ETM). The condition code and location are set up from the initial TSW. If a start address is given (SVC6.SAD), it is moved to TCB.CPSW+2. The start address is validated and a carriage return, specifying no starting parameters, is placed at UTOP. If delay start is specified, the start address followed by an SVC 2 instruction, a branch to register 8, and a SVC 2 code 23 parameter block are placed in the R8 through

R15 save areas of the TCB (TCB.RSAV+16) and TCB.CPSW+2 is set to point to the SVC 2 instruction at TCB.RSAV+18. Error code 10(A) is returned if the time field (SVC6.TIM) is invalid. The dormant and console wait bits are reset in TCB.STAT, and SVC6.TST is set to this updated status.

- LOAD (SVC6.LT) - The ESL state is entered. The LU specified is placed in an SVC 1 parameter block (LT.TSV1). The module's LIB is read into TCB.RSAV, effectively reading it into the registers. PAR.FND0 is called to find the partition the task is established for. Error code 76(4C) is returned if the partition does not exist. If the partition is vacant, the loader part 3 (LT.PA) is called in to perform the load. Otherwise, in order to load the task, a roll-out must occur. Error code 73(49) is returned if the task is not rollable because of any of the following conditions:

    1. OPTION ROLL not set.
    2. Priority is not less than the priority of the task to be loaded.
    3. Task is connected to a TGD.
    4. Task has an outstanding time interval.

  If the task is rollable, it is removed from the following waits:

    1. Dormant
    2. Console wait
    3. Any SVC wait

  The roll pending flag is set (TFLG.ROM), the task's previous status and current priority and the caller's TCB pointer are

saved in otherwise unused areas of the TCB. The task's priority is set equal to the priority of the task requesting the load. The loader then puts itself (as a subroutine of the calling task) into SVC 6 wait. This wait can be removed in two ways:

1. The scheduler (SCHED) discovers that there are no outstanding I/O requests initiated by the task to be rolled.
2. The task to be rolled is cancelled or goes to end of task (SVC3) and no longer has to be rolled out.

If the task does not go through SVC3, the loader continues execution by calling in the second part (LT.CS2) to perform the roll. Otherwise, the third part is entered at LT.CL to perform the load.

LT.CS2 uses the load LU (SVC6.LU) to assign the roll file by saving the contents of the logical unit table entry (TCB.LTAB) and the attribute byte table entry. The LU table entry is then set to zero and the roll file is assigned. The TCB, followed by the contents of the partition, are written to the roll file. The rolled task is set dormant.

If the background is not being rolled, the TCB.ID and the command processor current task are reset. The roll out bit (bit 15) is set in the corresponding partition table entry for the rolled task. The roll file is closed and the TCB.LTAB entry is restored to its previous contents. All logical units for the rolled task are zeroed and the third part of the loader is enter at LT.CL. Possible errors returned by the second part of the loader are:

75(4B) Assign or close error on the roll file

74(4A) I/O error on the roll file

LT.PA checks that the task fits in the partition. If not, error code 76(4C) is returned. If the task is to be loaded into the background partition, and the directed task is not '.BG', error code 1 is returned. For a foreground task, error code 1 is returned if the name does not conform to OS/16 naming conventions.

If the task uses a library or task common, PAR.FND is called to ensure that the partitions exist. If not, error code 66(42) is returned. LOAD.MOD is called to load the task, and CHKSUM is called to verify a good checksum. Error code 18(12) is returned for a checksum error.

If, for any of the above reasons, the load was unsuccessful the rolled-out task is rolled back in.

The initial TSW/LOC are set up from the first four bytes of the UDL. The maximum and current priorities are set up in the TCB and in SVC6.RPI. The task name (TCB.ID) is set up from SVC6.ID. For a task, UDL.UBOT, UDL.CTOP, UDL.CBOT and UDL.UTOP are set up. PLINK is called to adjust the priority table. TCB.OPT is set to the options in LIB.OPT. R5, R6, and R7 are set up to their proper contents for SVC 6 function determination (SVC6.IF) and a normal exit is made.

5.2.6 SVC 9 Executor (SVC9)

Entry to SVC9 is in the SU state. The caller's general and floating point registers are saved in TCB.RSAV and UDL.FPRS, respectively. The updated PSW is saved in TCB.CPSW. The requested new TSW is loaded from the SVC instruction's effective address. If the new TSW has task queue service traps enabled, the task queue is checked. If it contains any entries, a trap is taken by saving the new TSW in UDL.TSKO and SVC9 continues with UDL.TSKN as the new TSE.

The new TSW is stored in TCB.CTSW. The condition code is extracted and is set in TCB.CPSW. The new LOC is checked by a call to ADD.BC and is stored in TCB.CPSW+2. Exit is made to IOTERM.

5.2.7 Address Check (ADDCHK)

The SVC routines call an entry point of ADDCHK to validate all addresses passed in the SVC parameter block. No checking is done for E-Tasks, nor for tasks built into a system with no Command Processor Module. If the task is in the background partition, the address is checked for the range of the background partition. For a foreground task, the address is checked for the range of the foreground area.

5.3 TASK TRAPS

5.3.1 Add to Task Queue (ADTSKQ)

ADTSKQ is called to queue a parameter to a task. R14 is set to a reason code as follows:

0	Device interrupt
1	Task request to queue parameter

2	Time interval expiration
3	I/O proceed complete
4	-Reserved-
5	Send message

Based on the reason code, the proper TSW enable bit is checked in TCB.CTSW. The queue address is picked up from UDL.TSKQ, the address is checked, and the parameter is added to the queue. If the queue service trap enable bit is set in TCB.CTSW, a task trap is to occur. If the task is in system level code, the trap-pending flag is set, so that the TSW swap occurs when the task returns to the UT/ET state. Otherwise, the trap wait bit (TSTT.TRM) is reset in TCB.STAT and TSW.SWP is called to swap task queue service TSWs.

5.3.2 TSW Swap (TSW.SWP)

TSW.SWP moves the current TSW/LOC to the old TSW/LOC in the UDL and the UDL's new TSW/LOC to the current TSW. The condition code from the current TSW is moved to the old TSW and the condition code from the new TSW is moved to the current PSW.

5.4 TIMER MANAGER

OS/16 MT2 optionally supports a Line Frequency Clock (LFC) and a Precision Interval Clock (PIC). System initialization enables the LFC, whereas the PIC is only enabled when a task requests a time wait or trap. The time of day is kept in ASCII hours: minutes:seconds and in binary seconds since midnight. The current date is kept in ASCII month/day/year or day/month/year depending on the DATE SYSGEN parameter.

All tasks requesting time waits or time traps are maintained on a timer service table (TIMTSV). There are enough entries in TIMTSV for every task SYSGENed into the system to have both types of time intervals outstanding. Each entry is one byte and normally contains zero. A used entry contains the TCB index of the task requesting the interval. The TCB index is made odd if the request is for a time trap (queue entry), and is even for a wait.

5.4.1 Handle LFC Interrupts (CLOCKISR), Clock ISR

The LFC interrupts at twice the line frequency. For example, if the line frequency is 60 Hz, the LFC interrupts 120 times per second (every 8 1/3 ms).

The clock ISR keeps a count which is initialized to twice the line frequency minus one. On every interrupt, the count is decremented by one. When it becomes minus (which means a second has elapsed), the clock update routine is scheduled and the count is reset to its original content.

5.4.2 Handle PIC Interrupts (TIMER), Timer ISR

When a time interval is outstanding, the PIC is set to interrupt at the SYSGENed INTERVAL.
The timer ISR schedules the timer update routine.

5.4.3 Clock Update Routine (CLKUPD)

The clock update routine is scheduled every second by the clock ISR. The binary time is incremented by one second.

If it is midnight (binary time = Y'00015180'), the binary time is reset to zero, the ASCII time is set to C'00:00:00', and the ASCII date is incremented by one day. The ASCII month and year are not changed by this routine and must be corrected by the SET TIME command every month.

If it is not midnight, the ASCII time is incremented by one second. The minute and hour are incremented when necessary.

After the time is updated, the time-out field in the DCB of every device in the DMT is decremented by one. If a device has timed-out, the current ISR is entered by a SINT on the device number. The clock update routine then exits to IOTERM.

5.4.4 Timer Update Routine (TIMUPD)

The timer update routine is scheduled at the SYSGENed INTERVAL by the timer ISR. Each task in the timer service table gets its TCB.TVAL or its TCB.ADQT decremented, depending on the low order bit of the timer service table entry. If this value becomes negative, the task is either taken out of time wait or has the timer parameter (TCB.ADQP) added to its task queue, depending on the type of interval. TIMCAN.2 is called to remove the table entry and move all of the other entries up one slot. If there are any more entries, the routine continues to decrement the TCB timer values. If all entries are removed from the table, the timer is disarmed. Exit is made to IOTERM.

5.4.5 SVC 2 Timer Calls

SVC 2 code 8, 9, and 23 deal with the clock routines. SVC 2 code 8 and 9 fetch the current time and date respectively. SVC 2 code 23 sets up for a time of day wait, a time of day trap, an interval wait, an interval trap, or cancels an outstanding time trap.

When a task requests a time of day trap or a time of day wait, the time specified is converted from seconds past midnight into millisecond from now and the time of day flag, used by the SET TIME command handler, is set in TCB.FLGS. The call is then processed as an interval request.

For an interval wait request, the time is moved to TCB.TVAL. For an interval trap request, the time is moved to TCB.ADQT, and the parameter is moved from the specified register (in the register save area) to TCB.ADQP.

The TCB index (for a wait) or the TCB index plus one (for a trap) is added to the timer service table. The PIC is enabled. For a wait, the task is put into time wait. Exit is made to SVC2RTN.

For a cancel interval request, TIMCAN is called to remove the entry containing the TCB index plus one. The user's condition code is set either to 0 or to 4 depending on the existence of an interval. Exit is made to SVC2RTN.

5.5 SYSTEM JOURNAL

OS/16 MT2 provides a facility for recording significant events in the system in a system journal. The journal is a standard circular list with the number of slots equal to five times the SYSGENed number of journal entries. The address of the journal is kept in the system pointer table (at SPT.JRNL) and is a system

entry point (JOURNL). Journal entries are made from system routines by executing a BAL instruction to the journal routine (JOURNAL), followed by a halfword journal code. Entry to the journal routine must be in the SU state. Each entry in the journal consists of five halfwords of information. (See Figure 5-3.)

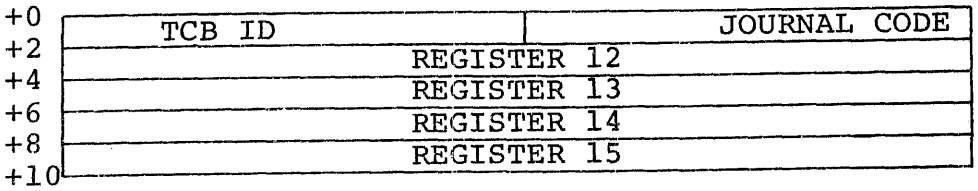


Figure 5-3. Journal Entry

The TCB ID is the TCB number of the current task at the time of the journal call. The last four halfwords are the contents of register 12 through 15 at the time of the journal call. When the journal list becomes full, the journal routine resets the number of slots used field and re-uses the list, thus maintaining the most recent entries. For a complete list of the journal codes made by the system see Appendix 1. For a description of how an executive task can make journal entries see Chapter 8.

5.6 SYSTEM MESSAGES

Since the executive routines cannot issue SVC calls, all messages output by the executive are queued to the command processor. This

is accomplished by the executive message subroutine, ATEXT.

A free log queue buffer is searched for. If none is found, the log message lost flag is set in CPS.FLGS. Otherwise, the task name is moved into the buffer, followed by the message. ADTSKQ is called to queue the message.

5.7 CRASH HANDLER

Throughout OS/16 MT2, checks are made for normally impossible states of the system such as an invalid item on the command processor's queue or an illegal instruction in system code. When such a condition is found, the system brings itself to a halt before further destroying the conditions that led up to the impossible situation. This is done by entering the crash handler.

The crash handler is entered by issuing a SINT instruction to device zero followed by a halfword crash code. The first entry in the ISP table is set by system initialization to branch to the crash handler, CRASHX. CRASHX saves all the registers in the crash save area, CRSHSV. The old PSW is checked to make sure entry was because of a SINT instruction. If not, entry was because of a hardware failure, and the crash code is set to 162. For a SINT the crash code is picked up from the halfword following the SINT instruction. The crash code is stored in SPT.CRSH. Register 5 is set to the address of the system journal, and register 6 is set to the address of the last entry made in the system journal. The crash code is displayed on the display panel and a

PSW is loaded with only the wait and machine malfunction enable bits set, thus stopping the system in an uninterruptable state. See Appendix 1 for a complete list of crash codes and their meanings.

5.8 INTERNAL INTERRUPT HANDLERS

The internal interrupt handlers process the interrupts generated by the microcode for illegal instruction, arithmetic fault, memory protect fault, system queue service, memory parity error, power fail, and power restore. In addition, illegal SVC calls and invalid addresses passed in SVC calls are processed.

Most internal interrupts are processed by the internal interrupt handler, EXEC. Entry to EXEC is in the SU state with the condition code specifying the type of interrupt (see Table 5-4).

EXEC saves the general registers and calls EXECSUB to save the floating point registers. A branch is made to the proper service routine.

TABLE 5-4. INTERNAL INTERRUPTS

CC	Internal Interrupt
0	Illegal Instruction
2	Floating Point Fault
4	Illegal SVC
6	System Queue Service
8	Fixed Point Divide Fault
A	Memory Protect Fault

5.8.1 Machine Malfunction Handler (MMH)

On detection of a memory parity error, a power fail, or a power restore, the machine malfunction handler is entered. Entry is directly from the microcode with all interrupts masked off. The condition code is used to determine the type of interrupt and the appropriate routine is entered.

On a memory parity error, the common error handler (IIMPH) is entered in the SU state.

For power fail detect, MMH tests an internal flag (PFFLAG) to see if a power restore sequence was in execution at the time of the power fail. If this is so, MMH simply loads an enabled wait PSW to wait for the power restore interrupt. If a power restore sequence was not in execution, the power fail flag is set. Since the list instruction traps are used by the power restore routine, and they are non-reentrant, the PSW location is checked for being within the list instruction traps. If in the traps, the traps are allowed to complete and are set to return back to the power fail routine at MMPF.4. Otherwise, the machine malfunction old PSW and the general registers are saved. The power restore auto/restart save areas are not used since multiple power fails would destroy the original state of the system. MMH then loads an enabled wait PSW to wait for the power restore interrupt.

On power restore detect, MMH enters the IS state and checks the power fail flag. If reset, the memory parity error handler is entered. Otherwise, the power restore message is written to the console device and a wait PSW is loaded. When the operator has reset all devices and depresses the RUN switch on the Processor, execution continues. All busy devices have their driver's abort routine entered.

Each task is checked for power fail trap enable. If enabled, the power fail trap pending flag (TFLG.PRM) is set. Otherwise, the pause pending flag (TFLG.PPM) is set. The task is removed from all waits except roll wait and I/O wait. The timer service table is cleared.

The system clocks are enabled, and a zero is added to the system queue to force an interrupt and enter the scheduler.

MMH then reloads the registers from the save area, resets the power fail flag, and reloads the machine malfunction old PSW from the internal save area.

5.8.2 Illegal Instruction Handler (III)

After going through the required traps, EXEC is entered to save the current task's registers. Return is to IIINT in the SU state which enters the common error handler (IIMPH).

5.8.3 Memory Protect Fault Handler (MPDVR)

MPDVR is entered on an interrupt from the memory protect controller. EXEC is entered to save the current task's registers. Return is to MPINT in the SU state. If the task is attempting to modify the SVC 13 or SVC 14 vector table entry (used by OS AIDS), it is allowed to. Otherwise, the common error handler (IIMPH) is entered.

5.8.4 Arithmetic Fault Handler (ARITHF)

The arithmetic fault handler is entered from EXEC at FPINT for a floating point fault, or at DVINT for a fixed point divide fault. The common error handler is entered at IMPH.

5.8.5 Illegal SVC Handler (SVCIL)

SVCIL is entered in the SU state from EXEC because of an illegal SVC number. SVC2IL is entered in the ESL state because of an illegal SVC 2 code. SVC6IL is entered in the ESL state because of an illegal SVC 5 option, or an SVC 6 from the background partition with OPTION SVCP (Illegal SVC 6 pause). The common error handler (IIMPH) is entered in the SU state.

5.8.6 Invalid SVC Address Handler (SVCILA)

SVCILA is entered in the SU state from SVC1 for one of the following reasons:

- The parameter block is not on a halfword boundary.
- For a U-Task, the parameter block is outside of the task's address space.
- The end address is less than the start address.

- For a U-Task, the start address or the end address is outside of the task's address space.
- The start address is not on a halfword boundary.

SVCILA is entered in the SU state from SVC9 for one of the following reasons:

- The parameter block is not on a halfword boundary.
- For a U-Task, the TSW to be loaded is outside of the task's address space.
- For a U-Task, the location specified by the TSW is outside of the task's address space.

SVC2LA is entered in the ESL state from SVC2 for one of the following reasons:

- The parameter block is not on a halfword boundary.
- For a U-Task, the parameter block is outside of the task's address space.
- For a U-Task, an address specified by the parameter block is outside of the task's address space.

SVC6LA is entered in the ESL state from SVC6 for one of the following reasons:

- The parameter block is not on a halfword boundary.
- For a U-Task, the parameter block is outside of the task's address space.
- For a called U-Task, the start address specified on an SVC6 start request is outside of the called task's address space.

SVC7LA is entered in the ESL state from SVC7 for one of the followirg reasons:

- The parameter block is not on a halfword boundary.
- For a U-Task, the parameter block is outside of the task's address space.

The invalid SVC address handler enters the common error handler (IIMPH in the SU state.

5.8.7 System Queue Service Handler (QTERM)

QTERM is entered in the SU state from EXEC. It moves the old PSW to the current task's TCB and exits to IOTERM.

5.8.8 Common Error Handler (IIMPH)

The common error handler is called to handle internal interrupts which may be generated by a task. Register 4 contains the address of a message parameter block (see Figure 5-4). Registers 14 and 15 contain the PSW at time of interrupt. If a task, other than the system task, was current at time of interrupt, the message is queued to the command processor and the task is paused. Otherwise, the crash handler is entered.

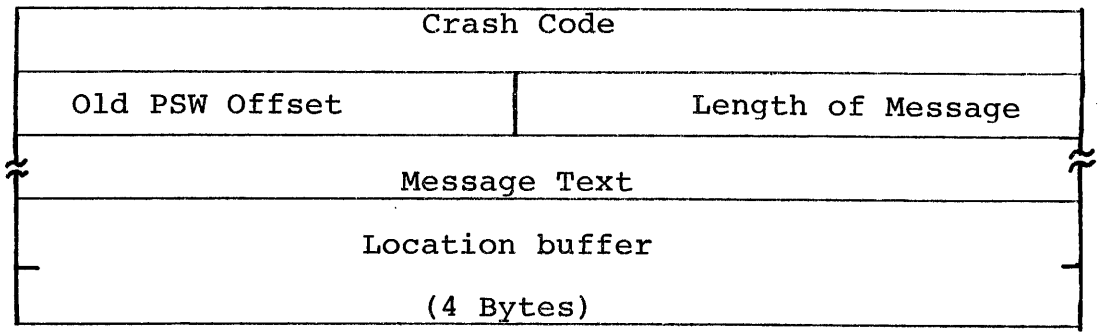


Figure 5-4 Common Error Parameter Block

CHAPTER 6
THE COMMAND PROCESSOR

6.1 INTRODUCTION

The command processor is the highest priority task in OS/16 MT2. It is an executive task (see Chapter 8). The command processor is the medium through which the console operator communicates with the operating system, and it controls the system environment. The command processor also controls the system console, memory partitions, and the Command Substitution System (CSS).

6.2 COMMAND MAIN

CMD.TOP is the main line routine of the command processor. It calls CMD.CKIO to ensure the console and current CSS level are active. The status fields of the two parameter blocks (CPS.CNIN and CPS.CSIN) define the current state of the console and the current CSS level as follows:

Status	Meaning
+2	needs a read (inactive)
+1	read is outstanding (active)
0	read has been completed successfully
-n	read has been completed with an error

CMD.CKIO checks to see if the console should request input from the operator (status=2). If so, it outputs the prompt '*' with a write/wait/image, sets the status to 1, and issues a read/proceed to LU 0. If CSS is in effect, CMD.CKIO checks the status of CPS.CSIN. If this status is 2, it checks if the background partition is currently dormant, and if so, sets the status to 1 and issues a read/proceed to the current CSS LU.

After calling CMD.CKIO, CMD.TOP checks the Command Processor Queue (CMDPQ). If the queue is empty, CMD.TOP issues an SVC 9 to put the command processor into trap wait with all interrupts enabled. If the queue is not empty, it takes an item off the queue and decodes it as follows:

Value	Routine	Meaning
CPS.CNIN	CMD.IN0	Command line from the console
CPS.CSIN	CMD.INX	Command line from current CSS level
CPS.IPRQ	CMD.TI	SVC 1 input request from a task
CPS.OPRQ	CTP.OUTP	SVC 1 output request from a task
CROLL	CMD.ROL	Roll-in request
A(BUFFER)+1	CTP.LOGR	Log message request
BSC.PARM	CMD.BSCT	Bulk storage command termination

6.3 COMMAND PARSING

CMD.IN0 or CMD.INX is entered when the I/O proceed termination trap for the console or the current CSS level respectively, has occurred. They check the parameter block status, and if the read

terminated properly, the status is set back to 2 (needs a read). Then CMD.NEXT is entered to process the line, unless one of the build commands is in effect.

CMD.NEXT is entered from CMD.IN and from all command executors to process the next command on the command line. Individual commands are moved from the buffer into which they were read to the buffer CMDBUF. If the command was entered from CSS, parameter substitution takes place each time an @ is found. When CMD.NEXT completes its moving, CMDBUF contains a single command with leading and trailing blanks deleted and a carriage return appended after the last character. The command mnemonic table is then scanned using an SVC 2 code 17. If the command was entered from CSS, a check is made to see if the skip to $TERMJOB flag (CFLG.SJ) is set. If so, only $JOB and $TERMJOB commands are processed; all others are ignored. If the skip to $ENDC flag (CFLG.SE) is set, only $IF, $ELSE, and $ENDC commands are processed. Processed commands read from CSS are then logged. If the command was in the command table, the appropriate executor is brought in and executed. Otherwise, unless the command starts with a '*', CMD.CSS is called to process a CSS call.

6.4 COMMAND PROCESSOR OVERLAYS

The command processor is overlayed in 16 parts. See Table 6-1 for the names of routines in each overlay. The command processor overlay loader (LDC) loads the required overlays.

TABLE 6-1. COMMAND PROCESSOR OVERLAYS

OVERLAY	ROUTINES
0	CMD.ROL CTP.OUTP CTP.LOGR CMD.IN CMD.TI CMD.BSCT
1	CMD.CSS CIN.NX (CMD.NEXT)
2	ERRMSG (ERROR)
3	$JOB $SKIP $IFx $ENDC $ELSE BUILD/ENDB $BUILD/$ENDB $COPY/$NOCOPY CSS.CLOS $CLEAR $EXIT $TERMJOB
4	START CANCEL CONTINUE PAUSE TASK OPTION SEND Bulk Storage
5	ASSIGN CLOSE ALLOCATE DELETE
6	BIAS EXAMINE MODIFY VOLUME MARK Part 2
7	MARK Part 1
8	RENAME REPROTECT CLEAR SAVE FILES Part 1
9	FILES Part 2
10	INITIALIZE
11	LDBG LFGR LOAD
12	SET LOG SET CODE SET PRIORITY SET TIME
13	SET PARTITION
14	DISPLAY LU DISPLAY MAP DISPLAY TIME
15	DISPLAY DEVICES DISPLAY PARAMETERS

6.5 COMMAND ERROR HANDLING (ERROR)

When an error occurs, and a command executor wants to output an error message, the ERROR routine is entered. The address of the error mnemonic must be contained in R14. For an error which gives no additional information other than the primary mnemonic (errors before ETAB.2 in the ERROR routine), R15 does not have to be set up. For errors between ETAB.2 and ETAB.4, R15 must

contain the address of a secondary mnemonic. The parameter block address must be contained in R15 for SVC 1, SVC 7, and SVC 6 type errors between ETAB.4 and ETAB.7. The error message is output by a call to CMD.LOG. CMD.TOP is then entered which, in effect, ignores the rest of the commands on the line.

6.6 COMMAND EXECUTORS

All command executors require that R2 contain a pointer to the command parameters, and R5 contain the TCB address of the currently selected task. In addition, the CSS '$' executors require that the Command Processor flags (CPS.FLGS) are contained in R7.

6.6.1 Task Related Commands

Certain commands pertain only to tasks. These commands are applied to the currently selected task as set by the TASK command.

The TASK command executor calls LOCT which scans all the TCBs in TCBTAB. If a task with a matching name is found, then its TCBTAB index is stored in CPT.CT or CPS.CCT, depending upon where the command was read from. If the task is entered as '.BG', the currently selected task is set to the background (index=4). If the system is SYSGENed without foreground partitions (FOREGRND 0), the current task is maintained as the background task, and the TASK command is illegal.

When a nonresident foreground task goes to End of Task (EOT), CLR.CT is called. If the task is the currently selected task, the current task is reset to zero (no current task). A task related command, entered when there is no currently selected task, causes a FUNC-ERR TASK.

6.6.2 Task Related Command Descriptions

The task related commands in OS/16 MT2 are as follows:

- START - Obtains the start location, sets up TCB.CPSW, moves the starting argument above UTOP, and sets TCB.STAT to zero.

- PAUSE - Sets pause pending in TCB.FLGS.

- CONTINUE - Resets the console wait bit in TCB.STAT.

- CANCEL - Changes to the SU state, calls CANCEL (in the Executive), changes back to the ET state.

- ASSIGN - Sets parameter block minus two (CPS.7T) to the TCB address of the current task for the file manager. Performs an SVC 7 assign.

- DISPLAY LU - Goes through TCB.LTAB. If the entry is zero, it skips to the next. If the LU is assigned to a device (address is below top of system), it calls DMTSRC to find the corresponding device name. If assigned to a file, FCB.VMT points to the volume name, FCB.NAME and FCB.EXT contain the filename and extension, respectively. The access privilege mnemonic is picked up from a table, based on the attribute byte in the TCB and the read and write counts in the DCB/FCB.

- CLOSE - Sets parameter block minus two (CPS.7T) to the TCB address of the current task for the file manager. Performs SVC 7 closes until all specified LUs are closed. Error status 9 (LU not assigned) is ignored.

- OPTIONS - Sets/resets the specified option bits in the user TCB option field (TCB.OPT).

- SET PRIORITY - Makes sure the specified priority is legal and not greater than the task's maximum priority. Stores the new priority in the UT/ET level priority byte.

- DISPLAY PARAMETERS - Unpacks and logs various parameters associated with the task. Parameters come from the task's TCB and UDL.

- SEND - The message is moved to CLOGB. An SVC 6 is issued to send the message.

6.6.3 Device and File Related Command Descriptions

The Device/File related commands are as follows:

- ALLOCATE - Executes an SVC 7 with data specified or defaulted.

- DELETE - Executes an SVC 7 with the file descriptor specified. Repeats for multiple FDs.

- RENAME - If the FD specified is a disc volume, it is renamed in the VMT and the VD. Otherwise, an SVC 7 assign for an ERW is issued followed by an SVC 7 rename.

- REPROTECT - Executes an SVC 7 to assign and reprotect the specified file/device.

- FILES - This executor is divided into two overlays. Overlay 8 contains the command parsing and the assigning of the disc and the print FD. Overlay 9 contains the directory search and file print logic.

- MARK - This executor is divided into two overlays. Overlay 7 contains the mark parsing, mark on, and mark overlay error handler. Overlay 6 contains mark off and the mark off/on roll handling. When the OS volume is marked off, overlay 7 remains in the overlay area and contains the logic necessary to mark it back on. While the OS volume is off, the only SVC 2 functions that can be used are SVC 2 code 17 (Scan), SVC 2 code 1 (Pause), and SVC 2 code 23 (Timer Management). SVC 7 assign is the only SVC 7 function that can be used.

  Mark packs the specified file descriptor with a call to MARK.PFD. DCB.CNTS are checked to ensure that no logical unit is assigned to this device (or to a file on this device). The write protect bit (DFL.WPM) is reset in DCB.FLGS so the device may be assigned with a write privilege. The device is assigned to LU 2 for ERW. DCB.FLGS are restored to the previous contents. Once the device is assigned, the 'OFF' or 'ON' mnemonics are scanned and the proper executor is entered.

  For Mark On, the 'PROTECT', 'ROLL', and 'OS' mnemonics are scanned and flags are set to note which ones are specified. If the device is a disc, MON.BULK is entered. Otherwise, the on-line flag is set in DCB.FLGS.

MON.BULK reads the volume descriptor. If the device was previously off-line, neither OS nor PROTECT are specified, and the on-line bit (VATR.ONM) is set in VD.ATRB, the mark is rejected with a STAT-ERR NOFF. Otherwise, the directory and bit map pointers are moved to the DCB. The volume name is validated. If OS is specified, the SVC 7 parameter block (CPS.S7) is set up to assign the OS file. If ROLL and PROTECT are specified, the mark is rejected with a PARM-ERR VAL. If ROLL is specified and any task is currently rolled out, the mark is rejected with a FUNC-ERR SEQ. The volume name is moved to the proper entry in the VMT. If OS is specified, the DCB counts are reset to SRW, an SVC 7 assign is issued to locate the OS file and the counts are reset to ERW. The command processor queue entries, which were saved on MOE.Q, are moved back to CMDPQ. A dummy SVC 2 and SVC 6 are executed to remove wait from all tasks which issued these SVCs while the OS volume was off. An attempt is made to re-write the VD with VATR.ONM set in VD.ATRB. If this attempt fails, the disc is hardware write protected so the software write protect flag (DFLG.WPM) is set. Overlay 6 is loaded and MON.6 is entered.

If ROLL is specified, roll files are deleted on the previous roll volume and new roll files are allocated. If ROLL is not specified, roll files are deleted if this volume was the roll volume. Overlay 7 is loaded and MON.MON is entered.

This routine outputs the mark on message and sets the on-line flag in the DCB. If OS is specified without PROTECT, the device was previously off-line and the on-line bit (VATR.ONM) is set in VD.ATRB, the message STAT-ERR NOFF is generated, but the volume is still on-line. Otherwise, exit is made to CMD.NEXT.

For Mark Off, overlay 6 is loaded and MARK.OFF is entered. If the device is not a disc, the on-line flag in DCB.FLGS is reset and exit is made to CMD.NEXT. Otherwise, MOF.BULK is entered.

The OS Mnemonic is scanned, and if found, a flag is set. If the volume to be marked off is the OS volume, OS must be specified. Otherwise, OS must not be specified. If this is the roll volume, the roll files are deleted (if no task is currently rolled out). The VMT entry and the bit-map presence flag are reset.

If the OS volume is being marked, all logical units are closed. Overlay 7 is then loaded and MOF.OFF7 is entered. The SVC 7 assign and the SVC 2 scan overlays are brought in. The OS file is closed by resetting OSOVDB and OSOVFD.

The on-line and write protect flags are reset in DCB.FLGS. The on-line bit (VATR.ONM) is reset in the volume descriptor of the disc. Exit is made to CMD.NEXT.

For Mark Overlay Error Handler, this routine is entered at LDC.R when the command processor-overlay loader discovers that there is no OS file from which to read in overlays. It decides which overlay load was attempted and handles the three cases itself.

If overlay 0 was to be loaded, R14 is checked. If it is not a command from the console, the queue item is saved on MOE.Q. If it is a command input termination from the console, MOE.IN0 is entered and it performs the functions as CMD.IN.

If overlay 1 was to be loaded, MOE.NEXT is entered which performs the same functions as CMD.NEXT.

If any other overlay was to be loaded, MOE.ERR is entered to log OVLY-ERR on the log device.

- DISPLAY DEVICES - Displays a list of all devices in the DMT, their device numbers, and their keys. It indicates whether the device is off-line or write protected. If the device is an on-line disc, it displays the name of the volume currently mounted on the device.

- FRECORD, FFILE, BRECORD, BFILE, WFILE, REWIND, RW - Picks up the proper SVC 1 function code byte and packs the FD. If LU is not specified, assigns the fd and issues an SVC 1 of the requested command. If LU is specified, the FCB address is obtained from the TCB of the current task. The fd specified is compared against the fd in the FCB. If the names match, the FCB counts are checked to ensure that the file is not assigned for any exclusive access. The file is 'patch assigned' to command processor LU 2 by storing the FCB address in the command processor TCB.LTAB. An SVC 1 is then issued to perform the requested function. Command processor LU 2 is then zeroed. An I/O proceed completion

trap occurs upon termination of the requested function. CMD.BSCT is entered when the trap occurs. The parameter block status is checked and ERROR is entered if necessary.

- INITIALIZE - The specified device is assigned for ERW. A check is made to ensure that the device is an off-line disc. Sector zero is readchecked to ensure that it is good. The size of the bit map is calculated by dividing the number of sectors on the device (DCB.SIZE) by 2048. A sufficient number of good sectors are found to contain the bit map. VD.MAP is set up in the VD buffer (INI.VD). The bit map is then zeroed. Sector 0 and the sectors occupied by the bit map are allocated in the bit map by issuing SVC 7s with function X'FF' and modifier X'01' (see Chapter 7). If readcheck is specified, all sectors within each cylinder are read by issuing reads to every Nth sector, where N is picked up from DCB.OINC. A cylinder of empty directory blocks is then allocated. On a 40 MB disc, three tracks of directory are allocated. On a 67 MB, 256 MB or on a floppy disc, one track is allocated. These blocks are allocated by units of N for maximum directory search efficiency. VD.FDP is set up in the VD buffer. The VD is then written to sector 0 and the bit map is updated by an SVC 7 checkpoint specifying LU X'FF' (see Chapter 7). Exit is made to CMD.NEXT.

- SAVE - Packs the fd. If the file does not exist, it allocates a contiguous file large enough to hold the image. An SVC 1 is issued to write out the OS.

- CLEAR - Packs the fd. Issues an SVC 7 delete of the specified volume (see Chapter 7).

6.6.4 General System Command Descriptions

The general system commands are as follows:

- BIAS - Stores bias value in BVALUE to be used by EXAMINE and MODIFY.
- EXAMINE - Gets the starting location and adds the bias. Checks for '/' or ','. If '/' is found, it gets the ending location, adds the bias, and computes the number of halfwords to be displayed. If ',' is found, it gets the number of halfwords to be displayed. The contents of memory, from the starting location through the ending location inclusive, are displayed, eight halfwords per line, to the log device.
- MODIFY - Obtains the starting address and adds the bias. Data obtained from the command line is stored in successive memory locations.
- SET LOG - Packs the fd and assigns it to LU 1. If no fd is specified, LU 1 is closed. If the copy option is specified, the copy flag (CFLG.LWC) is set in CPS.FLGS.
- VOLUME - Sets up the system default volume (SYSVOL). If no volume is specified, the system default volume (SYSVOL), the roll volume (ROLVOL), and the OS overlay fd (OSOVFD) are displayed.
- SET TIME - Scans the input line and picks up the date in the form mm/dd/yy or dd/mm/yy depending on the DATE SYSGEN parameter. It determines the validity of the date. The date is then stored in CALDAT. The time is obtained in the form hh:mm:ss, and is checked for validity. It is converted to seconds since midnight. If

there are any items on the timer service table that are in time of day wait or time of day add to queue, the corresponding TCBs are updated to reflect the change in time. The ASCII time is stored in CBUF and the binary time in SECPM.

- DISPLAY TIME - The date and time are obtained using an SVC 2 code 8 and code 9 and are displayed.
- DISPLAY MAP - For all partitions, the partition number, name (if a task is loaded in the partition), starting address, status, and priority (if a task is loaded in the partition) are displayed. The address of .SYS, FBOT, and MTOP are also displayed.
- SET PARTITION - The current partition table (PARTAB) is copied to the work buffer (SPBUF). The partition modify flag byte for each partition (MKBUF) is set to zero. Each partition address specified is stored in the SPBUF entry corresponding to the specified partition number. The corresponding MKBUF entry is set to the partition number. After all parameters are parsed, each SPBUF entry is compared to the next to check if the partitions are still in order. If a partition address is found which is not in order, the corresponding MKBUF entry is checked. If the MKBUF entry contains other than the partition number (partition not specified by user), the out of order partition address is made equal to the previous partition address and MKBUF is set indicating a change was made. The loop then continues. If the MKBUF entry contains the partition number (partition specified by user), a backwards scan is started. If the preceding SPBUF entry is not in order, the corresponding MKBUF entry is checked. If set to the part-

ition number, this partition was also specified by the user, so a STAT-ERR VAL is given. Otherwise, the preceeding SPBUF entry is set equal to the current SPBUF entry, MKBUF is set, and the loop continues. After the backward scan finishes, the forward scan continues. After the forward loop terminates, all changed partitions are checked to ensure that they are vacant and the partition's UDLs are set up. In a system with memory protect support, the address of partitions 1 and .SYS are checked for a 1K boundary. SPBUF is then copied to PARTAB. The memory protect pattern is then set up with a call to MEMPT. New roll files are allocated for changed partitions. Exit is made to CMD.NEXT.

6.6.5 Load Command Descriptions

- LDBG - Ensures that the background is dormant, assigns fd for SRO, and loads standard object records. When an end of program loader item is read, it sets up TCB.OPT floating point, memory resident, and background bits. It sets up UDL.UBOT, UBL.UTOP, UDL.CBOT, and UDL.CTOP. Then it exits to CMD.NEXT.
- LFGR - Ensures that the specified foreground partition exists and is vacant, assigns for SRO, and loads standard object records. When an end of program loader item is read, it sets up TCB.OPT floating point and memory resident. It sets up TCB.ID, UDL.UBOT, UDL.UTOP, UDL.CBOT, and UDL.CTOP. Then it exists to CMD.NEXT.
- LOAD - Sets up an SVC 6 parameter block, assigns fd for SRO, and issues an SVC 6 to load the task, reentrant library, or task common block.

6.7 COMMAND SUBSTITUTION SYSTEM (CSS)

The Command Substitution System (CSS) is a means for the user to create cataloged, but dynamically variable, command input streams to perform a predefined job.

6.7.1 Calling CSS (CMD.CSS)

Whenever a command is parsed, and it is determined that the command is not in the table of command mnemonics, it is then assumed that a CSS call is being made. The mnemonic is treated as a file descriptor, and an attempt is made to assign it. If the fd does not have an extension, then '.CSS' is appended. If the file/device does not exist, a MNEM-ERR message is issued.

Since the command executors use LUs 0 to 4, CSS files are assigned starting at LU 5 (level 1 = LU 5, level 2 = LU 6, etc.). The calling command string is moved from CMDBUF to the corresponding buffer for the called CSS level in the CNSLBUFF buffer pool. The SVC 1 parameter block (CPS.CSIN) is then set up to read into the next buffer in the pool. The status is set to 2 so that CMD.CKIO issues a read to the LU. The current CSS level (CPS.CLEV) is incremented by one. If the call was from the console, the current CSS task (CPS.CCT) is set equal to the current task (CPS.CT). CMD.CSS exits to CMD.TOP

6.7.2 Parameter Substitution (CSS.SUBS)

If an @ is found while moving the input string to CMDBUF, the actual parameter is substituted for the parameter reference. The parameter number and the address of the appropriate parameter are obtained. The parameter (if it exists) is moved to CMDBUF. If, in moving the parameter into CMDBUF, the CMDBUF pointer exceeds the size of CMDBUF (CMDLEN+2),

a CSS-ERR BUFF is generated. If the parameter number refers to a non-existent parameter, no characters are moved into CMDBUF and the routine completes normally.

6.7.3 Additional CSS Commands

Several additional commands are supplied (only when the CSS SYSGEN parameter is specified) to allow the user greater flexibility in creating CSS files and testing conditions. They are as follows:

- $COPY and $NOCOPY - These commands turn on ($COPY) or off ($NOCOPY) the display of CSS commands executed from a CSS file. They are or are not logged, depending on whether $COPY or $NOCOPY is in effect. These executors merely set or reset a flag (CLFG.DC) in CPS.FLGS used by CMD.NEXT to determine whether to log the command.
- $CLEAR - This command terminates all CSS processing, closes all CSS LUs, and resets all CSS flags.
- $JOB - Sets $JOB mode (CFLG.JM) in CPS.FLGS, saves the CSS level number that the $JOB appears on in CPS.JLEV. If there is a current task, its return code (TCB.RC) is set to zero. The current $IF level is reset to zero.
- $TERMJOB - Resets all CSS flags, resets the $IF level to zero, and closes CSS LUs down to the $JOB level.
- $SKIP - If there is a current task, its return code (TCB.RC) is set to 255. If $JOB is in effect, the skip to $TERMJOB flag (CFLG.SJ) is set in CPS.FLGS. The $IF level is reset to zero and the CSS LUs are closed down to the $JOB level. If $JOB is not in effect, the $CLEAR executor is entered.
- $EXIT - The current CSS LU is closed and input begins from the previous level. If the $JOB level is greater than the previous

level, it is set to the previous level. The current $IF level is reset to zero.

- $IFE, $IFNE, $IFG, $IFNG, $IFL, $IFNL - These commands get the value specified in the operand field of the command, and compare it to the current return code (TCB.RC). The $IF level (IFLV) is incremented by one. If the compare satisfies the condition specified, exit is made to CMD.NEXT. Otherwise, the skip to $ENDC flag (CFLG.SE) is set in CPS.FLGS and the skip level (SKLV) is set equal to the $IF level (IFLV).
- $IFX, $IFNX - Same as previous $IFs except, instead of the compare, an assign attempt is made on the specified file. Success indicates it exists, but certain errors also indicate the file exists.
- $IFNULL, $IFNNULL - Same as previous $IFs except, instead of the compare, the next character is checked. If it is a CR, the parameter is null. Otherwise, it is not null.
- $ELSE - If the skip to $ENDC flag (CFLG.SE) is reset, it is set and the skip level (SKLV) is set equal to the $IF level (IFLV). If set, the current $IF level is compared to the skip level. If not equal, exit is made to CMD.NEXT. If equal, the skip to $ENDC flag is reset and the skip level is set to zero. Exit is made to CMD.NEXT.
- $ENDC - The $IF level is decremented by one. If the skip to $ENDC flag is reset, exit is made to CMD.NEXT. If set, the skip level is compared to the previous $IF level. If not equal, exit is made to CMD.NEXT. If equal, the skip to $ENDC flag is reset and the skip level is set to zero. Exit is made to CMD.NEXT.
- SET CODE - The parameter is packed and stored in TCB.RC for the current task.

- BUILD/ENDB, $BUILD/$ENDB - The fd is packed and assigned to LU 4. If the file does not exist, it is allocated with CMDLEN record length and blocksize of 1/1. CFLG.BLD or CFLG.DB is set appropriately.

  Each time a line is read and CMD.IN is entered, these flags are checked. If either is set, CSS.BLD is entered.

  CSS.BLD checks the $BUILD flag (CFLG.DB) and enters CSS.DB if it is set. Otherwise, it checks for ENDB. If it is ENDB, the build flags are reset and exit is made to CMD.TOP. If other than ENDB, the line is written out to LU 4 and exit is made to CMD.TOP.

  CSS.DB checks for $ENDB. If found, the build flags are reset and exit is made to CMD.TOP. Otherwise, the line is preprocessed into CMDBUF, by performing parameter substitution. The line is written out to LU 4 and exit is made to CMD.TOP.

6.8 CONSOLE HANDLING

The system console device is controlled by the command processor. Any I/O requests issued to this device cause a queue entry to be added to the command processor's queue (CMDPQ).

The command processor always has the real console device assigned to its LU 0. If SET LOG is in effect, LU 1 is assigned to the SET LOG device. When the command processor wishes to output a message to the console/log device, CMD.LOG is called.

CMD.LOG checks if LU 1 is assigned. If so, the time followed by the message is moved into CLOGB and is output to LU 1. Then the log with copy flag (CFLG.LWC) is checked. If set, the message is output to LU 0.

If LU 1 is not assigned, the message is output to LU 0.

6.8.1 SVC 1 Output Requests

CTP.OUTP processes SVC 1 write requests to the console device. The task name, followed by the message, is moved into COUTB. CMD.LOG is called to output the text. The dummy termination routine (DMYTRM) is then scheduled.

6.8.2 SVC 1 Input Requests

CMD.TI processes SVC 1 read requests to the console device. The task name followed by a promt (>) is output with an SVC 1 to LU 0. Then an SVC 1 read is issued. The text read is moved into COUTB following the prompt. CMD.LOG is called to log the input. The dummy termination routine (DMYTRM) is then scheduled.

6.8.3 Log Message Requests

CTP.LOGR processes log messages to the console device. CMD.LOG is called to output the message. The log message buffer is freed by zeroing the first halfword.

6.8.4 The Break Key

The break key (or escape) on the console has special meaning to OS/16. It causes any I/O to be terminated. CMD.CERR, along with CPS.CENT, control the console break. When the console driver detects break or escape, it returns error status to the command processor. CMD.CERR is called in this case. Normally, CPS.CENT is set to zero in CMD.CKIO. When the task I/O routines are entered, CPS.CENT is set to the queue entry removed from CMDPQ.

CMD.CERR checks CPS.CENT. If non-zero, it adds the value to the bottom of CMDPQ. A time wait for ½ second is then issued. An SVC 1 is issued to output the '*' prompt. Then a read is issued to read a command line. After a command line is read, CMD.TOP or CMD.NEXT is entered, depending on whether there is an unfinished line from CSS.

6.8.5 Halt I/O

When an SVC 1 Halt I/O is issued to the console, the command processor takes appropriate action depending on how far the user I/O has progressed before it is halted.

If the Halt I/O is performed while the command processor is scheduling to output the TASKID, this I/O is aborted and the dummy termination routine is scheduled.

When the read is halted prior to the SVC 1 read to the real console, but after the TASKID has been logged on the console, the command processor schedules the Termination Routine for the dummy driver which terminates the I/O.

If the SVC 1 read to the real console has been issued at the time of the Halt I/O, the abort routine of the real console driver is entered to cancel the I/O.

Halt I/O status (X'8281') is returned in the Dummy Driver Termination routine (DMYTRM) if the Halt I/O abort flag (DFLG.HIM) is set in the Dummy DCB.

6.9 SYSTEM INITIALIZATION

System initialization is performed by the routine INIT. It is entered in the IS state whenever the system is started at X'60'. The system queues and flags are cleared. Top of memory is searched for and MTOP and FBOT are set up.

6.9.1 System with the Command Processor Module

The partition address table, PARTAB, is set up. All TCBs are set to initial values and the priority table, PRITAB, is set up. The DCBs and the VMT are set up. The ISPTAB is set up for the console device. The command processor buffers and flags are cleared. The system clocks are enabled and the command processor is set ready when the PSW pointing to INMARK. The scheduler, SCHED, is then entered in the SU state. INMARK is scheduled in the ET state. The disc with device number in location X'72' and the SELCH device number floppy selector in location X'70' is marked on-line with protect and the system start up message is logged. The command processor is started at CMD.TOP.

6.9.2 System with no Command Processor Module

On a system with no operator interface, the partition table and TCBs are set up by CUP/16 at system generation and are not initialized by the system initialization code. If the system has no SVC 7 support, CUP/16 also sets up the LU assignments as specified by the user.

If the system has no disc devices, the dummy command processor is set dormant and the scheduler (SCHED) is entered in the SU state.

If the system includes support for one or more disc devices, the dummy command processor (INMARK) is set ready, and the scheduler is entered in the SU state. INMARK is scheduled in the ET state. ALL discs configure in the system are marked on-line, with S/W protect if H/W protected and without S/W protect if not H/W protected as follows:

Command processor LU 2 is patch assigned to the disc by storing the DCB address of the Disc into the TCB.LTAB entry. The LU attributes are set equal to the Device attributes (DCB.ATRB). The Volume Description is ready by an SVC 1. The Directory Pointer (VD.FDP) is moved to DCB.DIRP and the Bit Map Pointer (VD.MAP) is moved to DCB.BITP. The VMT entry is found by a call to VMTSRC. The Volume Name is validated by a call to NAMCHK. VOLCHK is called to ensure that a duplicate volume/device name is not present. The Volume Name is moved to the proper VMT entry. If the on-line bit is set in VD.ATRB, DCB.FLGS are set for on-line and write protect. Otherwise, an attempt is made to re-write the VD with the on-line bit set. The on-line bit is set in DCB.FLGS, and if the write returned an error, the write protect bit is set. The patch assignment is closed by setting the TCB.LTAB entry to zero.

After all discs are marked on-line, INMARK is set permanently dormant. The scheduling of tasks continues as normal.

CHAPTER 7
FILE MANAGEMENT SYSTEM

7.1 FILE MANAGER

The routines in this program include all the logic needed to support the OS/16 MT2 File Management System. The file handler (SVC7) is invoked by the Supervisor Call Executive (SVCEXEC) any time a task issues an SVC 7 supervisor call and the file handler is not currently in use (roadblocked). When entered, SVC7 decodes each function specified by the SVC 7 parameter block, then loads and invokes the necessary executors The SVC 7 executors contain routine to:

- Allocate a new file
- Assign a file or device to a logical unit
- Change the access privileges of a logical unit assignment
- Rename a file or device
- Reprotect (change the protection keys of) a file or device
- Close the assignment between a logical unit and a file or device
- Delete a file
- Checkpoint a logical unit
- Fetch the attributes associated with a logical unit assignment

More than one function can be performed by a single SVC 7 request. Each executor that completes successfully returns to S7.CMD to determine if any other requests are still outstanding. When all functions have been processed, control is returned to the calling task and SVC 7 is un-roadblocked by SVC7RN. If any of the SVC 7 executors encounter an error, the appropriate error status is returned in the calling task's parameter block and control returns by way of SVC7RN. The executors make use of the resident routines contained within the file manager, such as:

Directory management routines for maintaining information on all currently allocated files.

Bit map management routines which provide a method for allocating and deleting files on direct-access volumes.

7.2 VOLUME ORGANIZATION AND INITIALIZATION

Any direct-access volume to be used within an OS/16 MT2 environment must be formatted by the Common Disc Formatter. Since OS/16 handles file allocations in multiples of one sector, the arguments to this program must specify DEFSEC 1. Once a volume has been formatted using this procedure, it should not have to be formatted again unless a hardware failure occurs on the volume. After a disc is formatted, it must be either initialized using the OS/16 INITIALIZE command or the OS/32 Disc Initialize Utility both of which optionally read-checks each sector on the volume. Any sector found to be defective is marked as permanently allocated. A bit map, directory, and volume descriptor (VD) are also written on the volume.

A volume descriptor is shown in Figure 7-1. The volume descriptor contains the volume name, the volume attribute bits, a pointer to the first directory block, and a pointer to the bit map.

VD.VOL Volume name	VD.ATRB Volume attributes	VD.FDP pointer to first directory block	VD.OSP (reserved)	VD.OSS (reserved)	VD.MAP pointer to bit map

Figure 7-1 Volume Descriptor

The size of the bit map is determined by the size of the volume. Each complete bit map sector represents 2048 allocatable sectors on the volume. The final sector within the bit map represents between 1 and 2048 sectors. A sector is marked as allocated when the bit representing

it is set (1) or free when the bit is reset (0).

The volume descriptor is placed on cylinder 0, track 0, sector 0. The bit map may be located anywhere on the volume since it is pointed to by the VD.

7.3 DIRECTORY MANAGEMENT

A file directory is maintained as a chain of directory blocks (see Figure 7-2). A chain field contains either a zero (indicating it is the last block in the chain) or the logical block address (sector) of the next block in the chain.

A volume that has just been INITIALIZED contains a directory of empty blocks. The blocks are chained together in a manner which decreases directory search time.

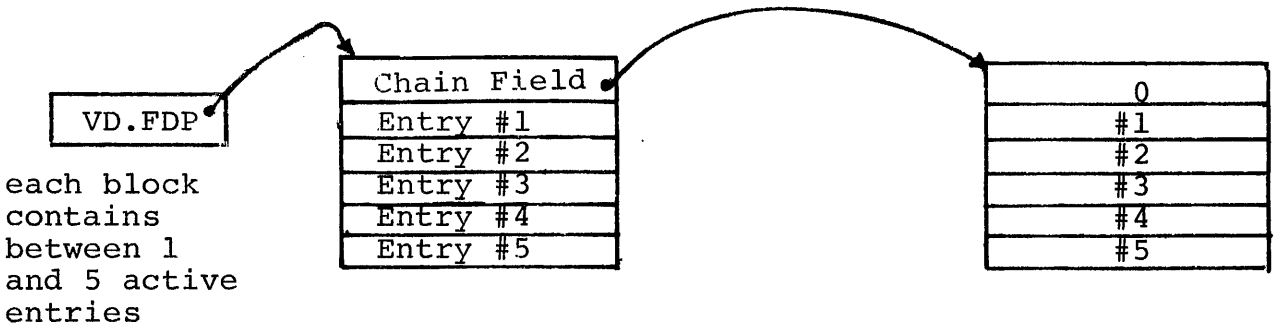


Figure 7-2 Directory Example

7.3.1 Directory Entry Creation (ALO.ALLD)

When the pre-allocated directory is filled with active entries, and a new file is allocated, a new directory block is allocated. The first entry represents the new file and the remaining four entries are marked inactive and therefore available for additional new files.

7.3.2 Directory Access (DIRLOOK,GETD,PUTD)

When a function is requested on a currently existing file, the directory block containing the Directory Entry (DIR) for the file must first be found. The I/O routines used to read directory blocks into memory or to write out modified blocks are GETD and PUTD.

When a new file is allocated, each block of the directory is searched until an inactive entry is found. If all entries are marked active, a new directory block is allocated as described previously.

7.4 BIT MAP MANAGEMENT

OS/16 direct access files are allocated in multiples of one sector. The status (free or allocated) of each sector on the volume is maintained in the volume's bit map. When a volume is INITIALIZED, all non-defective sectors within the volume are marked as free by resetting the corresponding bit in the bit map. The VD, bit map, and directory are then created. The sectors they occupy are marked as allocated by setting the appropriate bits. The INITIALIZE logic also provides a pointer from the VD to the bit map (VD.MAP), and to the directory (VD.FDP).

When a request is received by the bit map management routines to allocate a string of contiguous sectors, GETSEC searches the bit map for a corresponding number of bits that are reset, thus indicating available sectors. The search starts at the current index of the current bit map sector. CHECKB is called to check bits in the bit map. GETB is called, if allocations span bit map sector boundaries, to read bit map sectors in this manner. GETSEC then sets each bit in the bit map within this allocation. ALLOB is called to set bits in this manner. As bit map sectors are modified, they are written back to the disc by a call to PUTB.

When a file is deleted, the procedure is reserved by RELEB (a part of the DELETE overlay). Each bit representing the allocation is reset by multiple calls to FREEB, indicating the sector is again available. GETB and PUTB may again be invoked to read and write the bit map sectors.

7.5 FILE HANDLER (SVC7)

SVCEXEC transfers control to the SVC 7 handler routine SVC7, passing two arguments; the address of the calling task's TCB and the address of the calling task's SVC 7 parameter block. If the calling task is the command processor, SVC7 replaces the calling task's TCB address with the TCB address at the parameter block address minus two. This allows the command processor to perform SVC 7 assign and close for a task other than itself.

SVC7 processes the function code specified by the parameter block from left to right. If the function code is initially zero, the call is a fetch attribute. If the calling task is the command processor, and the function code is initially X'FF', the call is a bit map manipulation call (used by the INITIALIZE command). Otherwise, the function code is saved in CMDBYTE, and each SVC 7 function specified within it is performed by loading and branching the appropriate executor. Each executor that completes successfully returns control to SVC7. As each function is performed, the bit representing it in CMDBYTE is reset until every bit of CMDBYTE has been reset. Control then returns to the calling task by a branch to the SVC 7 unblock routine SVC7RN.

7.6 SVC 7 OVERLAYS

The file manager is overlayed in eight parts. See Table 7-1 for the names of routines in each overlay. The SVC 7 overlay loader (LDS7) loads the required overlays.

TABLE 7-1. FILE MANAGER OVERLAYS

OVERLAY	ROUTINES
0	S7ALLO
1	S7ASSIGN part 1
2	S7ASSIGN part 2 GETFCB ASN.FMLU
3	S7CAP
4	S7RENAME S7REPROT EXCHECK
5	S7CLOSE RELEFCB
6	S7DELETE RELEB
7	S7CHECKP S7FETCH S7.BMP

7.7 SVC 7 FUNCTION EXECUTORS

7.7.1 Allocate (S7ALLO)

The SVC 7 executor S7ALLO is called directly from SVC7 when the function code in the parameter block specifies an allocate operation.

The directory management routines are called to ensure that the specified file descriptor is unique to the file specified, and to establish a directory entry for the file being allocated. For a contiguous type file, the complete file allocation size is established at allocation time by the bit map management routines. Since an indexed file is open-ended and has no predefined size, no bit map allocations are performed on behalf of an indexed file at allocation time. The necessary initial information is established in the directory for both file types. Control returns directly to S7.CMD upon successful completion of S7ALLO.

7.7.2 Assign (S7ASSIGN,ASN.CO,ASN.IN)

The SVC 7 executor S7ASSIGN performs all common assign processing for all assign calls. S7ASSIGN establishes the validity of the logical unit being assigned with a call to LUCHECK.

If the assign function is being performed on a device, control is transferred to ASN.DMT. ASN.DMT completes any necessary validity checks using the integrity checking subroutines APCHECK and ASN.KYCK (described later), and sets up the entry in the LU table to contain the DCB address and the corresponding entry in the attribute table to contain the device attributes. If the assign function is being performed on a file, control is transferred to ASN.VMT which validates the file descriptor with a call to FDCHECK, searches the directory for the file, and checks access privileges and keys (APCHECK and ASN.KYCK). If the file being assigned is the OS overlay file, then only SRO access is permitted. If the command processor is assigning the OS overlay file, and the logical unit specified is X'FF', then it is a call from the MARK command executor to assign the OS file. In this case OSOVST and OSOVDB are set up to contain the file starting sector number and the device's DCB address respectively. Otherwise, the second part of assign is loaded into the overlay area and control is transferred to it (ASN.A). There, the file manager logical unit TCB.FMLU) is set up by a call to ASN.FMLU which increments DCB.CNTS as if the disc were being assigned for SRW access. The appropriate routine, ASN.CO or ASN.IN, depending on the file type, is then entered.

ASN.CO and ASN.IN call the system space management routine (GETFCB) to allocate a file control block (FCB) within dynamic system space. GETFCB also moves control information from the file's directory entry to the FCB.

For a contiguous file, control is passed to ASN.CO. If the file is being assigned for a Read privilege, FCB.CSEC is set to zero. ASN.UPDT is entered to finish.

For an indexed file, control is passed to ASN.IN. If the file being assigned contains no index blocks, ASN.IN calls the bit map management routines to allocate an index block and a data block for the file. If the file already has index blocks, the first index block and the first data block are read into the FCB buffers with an SVC 1 read/wait call to the file manager logical unit (X'FF'). If the file is being assigned for write only (SWO or EWO), it is positioned at the end by setting the current logical record number (FCB.CLRL) to the number of logical records (DIR.CSEC). Otherwise, the file is positioned at the beginning by setting FCB.CLRL to zero. ASN.UPDT is entered to finish the assign.

ASN.UPDT sets up the logical unit table with the file's FCB address, and the attribute table with the file's attributes. The directory is updated and written out by a call to PUTD. The routine then returns to S7.CMD.

7.7.3 Change Access Privileges (S7CAP)

The SVC 7 executor, S7CAP, changes the access privileges associated with a given logical unit which is assigned to a file or device.

S7CAP ensures that the new access privileges are legal, removes the old privileges with a call to RESET, and assigns the new privileges with a call to APCHECK. This requires modifying the write and read count fields in the DCB, or the FCB/DIR (DCB.WCNT/DCB.RCNT; FCB.WCNT/FCB.RCNT,DIR.WCNT/DIR.RCNT) to reflect the current access privileges. The access privileges associated with a file are reflected in the WCNT and RCNT fields of the control block in the following manner:

WCNT/RCNT = 0 implies no task having write/read privileges

WCNT/RCNT = -1 implies one task having exclusive write/read privileges

WCNT/RCNT = +n implies n tasks having shared write/read privileges

7.4.7 Rename (S7RENAME)

S7RENAME is the SVC 7 executor that changes the name of a file or device. S7RENAME ensures that the device or file is assigned for ERW by calling EXCHECK. If the rename function is directed at a device, REN.DCB ensures that the new name does not currently exist in the Device Mnemonic Table (DMT) or in the Volume Mnemonic Table (VMT). If the new name is unique, the device's previous name is replaced with its new name in the DMT.

To rename a file, the procedure is similar except that it is the directory that is checked for a duplicate name. The directory management routines are used to read the directory, search for a name match, and replace it with the new file name. S7RENAME returns to S7.CMD upon successful completion.

7.7.5 Reprotect (S7REPROT)

The SVC 7 executor, S7REPROT, modifies the protection keys associated with a given LU. The LU must be assigned to a file or device for ERW (EXCHECK). The keys associated with a device are kept in its DCB (DCB.WKEY,DCB.RKEY). The keys associated with a file are kept in its directory entry (DIR.WKEY, DIR.RKEY). A file or device may be unconditionally protected (keys = X'FF'), unconditionally unprotected (keys = X'00'), or conditionally protected with write/read keys between X'01' and X'FE'. The logic in S7REPROT ensures that the new protected keys are not in violation of the former protect keys and updates the control block (DCB or DIR) with the new keys. Control returns to S7.CMD for further SVC 7 processing.

7.7.6 Close (S7CLOSE)

The purpose of the S7CLOSE executor is to disconnect an assigned logical unit from a file or device. The logic of S7CLOSE ensures that the given LU is currently assigned. If the close call was not from the CLOSE command in the command processor, an SVC 1 wait only is performed on the logical unit to ensure that all ongoing I/O and proceed calls are complete. The LU entry in the LU table is set to zero.

If the LU was assigned to a device, the read and write count fields in the DCB are modified as follows:

If old count = -1, new count = 0
(previously one exclusive user)

If old count = +n, new count = n-1
(previously n shared users)

If the LU is assigned to a file, the directory block is read by GETD. If the file is indexed, RESET.IN and PUTB are called to ensure that all index and data blocks in memory are written and that the bit map on the disc is up to date. For either file type, the count fields in the directory are updated as specified above. The FCB's memory allocation is returned to system space by a call to the memory management routine, RELEFCB. The directory management routines are then used to update the directory with the information about the file which was in the FCB. S7CLOSE exits to S7.CMD to process the remaining function code bits. bits.

7.7.7 Delete (S7DELETE)

The S7DELETE executor deletes contiguous and indexed files. A file is deleted by releasing its allocated storage on the volume containing it

using the bit map management routines, and by relinquishing its directory entry by means of the directory management routines. The command processor CLEAR command deletes all files on the volume. It performs an SVC 7 delete with the volume name set up, but the filename is set to blanks. This signals the delete routine to perform a clear operation instead of just a delete. The command processor is the only task that may issue such a call. S7DELETE returns control to S7.CMD upon completion.

7.7.8 Checkpoint (S7CHECKP)

The S7CHECKP executor checkpoints a logical unit which is assigned to a file or a device. If the checkpoint function is directed to a device, an SVC 1 wait only operation is performed on the LU. To checkpoint a file, all current information about the file is moved from its FCB to its directory entry. The bit map and directory management routines are used to ensure that the bit map and directory on the volume reflect current file allocations. An indexed file is also checkpointed to read mode using the indexed file reset routine, RESET.IN. After a wait only is performed on the LU, S7CHECKP exits to S7.CMD.

If command processor performs a checkpoint of LU X'FF', a PUTB operation is performed instead. This is used in the INITIALIZE command handler to update the bit map.

7.7.9 Fetch Attributes (S7FETCH)

The purpose of the S7FETCH executor is to obtain the attributes associated with the file or device assigned to a given LU. The device/file attributes, device code, and name are moved from the DCB/FCB to the task's SVC 7 parameter block. S7FETCH returns directly to the

calling task by branching to S7.RTN.

7.8 SVC 7 MEMORY MANAGEMENT ROUTINES

This section describes the memory management routines used by the SVC 7 executors:

- GETFCB - Allocates a new FCB. FBOT is decremented by the size requested. If the new FBOT is below SYSBOT, an error is returned. The FCB is then set up with information from the directory.

- RELEFCB - Releases an FCB. If the FCB to be released is at FBOT, FBOT is incremented by the size of the FCB and the routine is exited. Otherwise, a flag is set telling the disc driver not to start any disc I/O operations, and any disc I/O that is already started is allowed to complete. Once all disc I/O is halted, lower FCBs are moved up to fill the gap left by the FCB released. All address constants within those FCBs are relocated to reflect the new buffer addresses. FCB addresses in the TCB logical unit table are relocated to reflect the new FCB addresses. Then disc I/O is allowed to continue.

7.9 SVC 7 INTEGRITY CHECKING SUBROUTINES

This section briefly describes the integrity checking subroutines used by the SVC 7 executors:

- APCHECK - Verifies the legality of the requested access privileges and converts the requested access privilege to a numeric quantity to be saved in the WCNT and RCNT field of the control block.

- LUCHECK - Determines if a given LU is assigned and picks up its entry from the LU table.

- FDCHECK - Checks the syntax of a given file or volume name.

- OSOVCK - Checks if the fd in the SVC 7 parameter block is the OS overlay fd.

7.10 SVC 7 SPECIAL EXECUTOR

If the command processor executes an SVC 7 call with all function bits set (CMD = X'FF'), the special executor S7.BMP is entered. The function of this executor is to manipulate bit map allocation. The SVC 7 parameter block is redefined somewhat for this call. The MOD fields contain the function. X'FF' means free a sector, X'00' means test a sector and X'01' means allocate a sector. The KYS field contains the DCB address of the device to which the call applies. The SIZ field contains the fullword sector address of the sector to be manipulated. The bit map manipulation routine, entered at MAN.COM, is called to perform the desired function. The status of a tested sector is returned in the STA field; 0 = free, 1 = allocated. Return is made to S7.RTN.

7.11 SVC 1 INTERCEPT ROUTINES

When the disc driver determines that an SVC 1 call is directed to a file, the file management SVC 1 intercept routine (MHDFCB) is entered to process the request.

7.11.1 Contiguous File Handler

The contiguous file handler consists of the following two routines:

- CONTIG - Processes data transfer requests to a contiguous file.

- CMD.CO - Processes command function requests to a contiguous file.

Initially, CONTIG is entered for both types of request. If the request

is a rewind, the current sector is zeroed and CONTIG exits to FMGR.IX1, the exit point if no physical I/O is performed, otherwise, I/O is to be performed. CHKBSY is called to set the device and the DCB busy. A test is made to see if the call is a command function, and if so, a branch to CMD.CO is taken.

7.11.2 Data Transfer for Contiguous Files (CONTIG)

The SVC 1 parameter block information is copied to the DCB. The random address is obtained either from the parameter block (for a random request) or from the FCB current sector pointer (for a sequential request). The random address is then relocated by adding the FLBA of the file into it. If the request is a read or a write, it is performed by calling the disc data transfer routine, DISCIO. If the request is a test and set, a read is performed by a call to DISCIO. The first halfword of the buffer is then tested. If zero, it is set to -1 and the first sector of the buffer is rewritten to the disc. If the first halfword is -1, the users condition code is set to X'F'. The routine exits by branching to CMD.OUT1 which takes the task out of I/O Wait before branching to FMGR.IEX because the proceed bit of the function code is ignored. (The task was put into I/O Wait even if the function code specified proceed.)

7.11.3 Command Requests to Contiguous Files (CMD.CO)

The command function intercept routine for contiguous files, CMD.CO contains five command executors. Each executor and its function is briefly described below:

- BACKSPACE RECORD (CMD.BSR) - Decrement FCB.CSEC by 1, read this sector to check for EOF.

- FORWARD SPACE RECORD (CMD.FSR) - Increment FCB.CSEC by 1, and proceed as in CMD.BSR.

- FORWARD SPACE FILE (CMD.FFM) - Issue reads starting at FCB.CSEC until a pseudo filemark is found.

- BACKSPACE FILE (CMD.BFM) - Same as forward space file, except that the filemark is searched for starting at FCB.CSEC and backing up one sector at a time.

- WRITE FILE MARK (CMD.WFM) - Increment FCB.CSEC by 1, write a pseudo filemark at that address.

7.11.4 Indexed File Handler

The indexed file handler consists of the following two routines:

- INDEXED - Process data transfer requests to an indexed file.
- CMD.IN - Process command requests to an indexed file.

7.11.5 Indexed File Handler Subroutines

The following is a brief description of the indexed file handler subroutines:

- INX.READ, INX.PORN, INX.FORL - Update the current index block offset (FCB.CINX). If the required data block pointer is contained in the next index block, the current index block is written (if it has been modified), and the next index block is read into memory. Entry at INX.PORN reads the previous or the next block, depending its argument. Entry at INX.FORL reads in the first index block or the last index block, depending its argument.

- CALC.NBK - Based upon a logical record number, compute the following: the data block containing the record, the offset of the record within that block, the index block containing this data

block pointer, and the offset within the index block of the data pointer.

- INX.GETL - Move a logical record from a system buffer to the task buffer. If the logical record spans physical blocks, a call is made to INX.GETP to read the next data block into memory.

- INX.PUTL - Move a logical record from the task buffer to the system buffer. If the logical record spans physical blocks, a check is performed. For a file being extended, the current data block is written and a new block is allocated by a call to INX.PUTP. For a file being updated, the current data block is written by a call to INX.PUTP and the next data block is read by a call to INX.GETP.

- INX.GETP - Perform physical writes to an indexed file. The current data block is written with a call to DISCIO. If the file is being extended, then FCB.CINX is incremented. If this causes an overflow of the current index block, a new index block is allocated through GETSEC. A pointer to the new index block is placed in the current index block and is written out. The new index block is zeroed and a pointer to the previous block is placed into it. The current index block offset (FCB.CINX) is set to eight to point to the first data pointer slot. The current index block number (FCB.CINB) and the number of index blocks (FCB.NINB) are incremented by one. When room is made for a data block pointer, a data block is allocated with a call to GETSEC. A pointer to this data block is placed in the index block. The current data block number (FCB.CBLK) and the total number of data blocks (FCB.NBLK) are incremented by one.

- RESET.IN - Change the current state of an indexed file. RESET.IN is called for each I/O operation to an indexed file to make sure that all current buffers are flushed, if necessary, prior to initiating a new I/O operation.

- INX.CUPD - Add one to current data block number (FCB.CBLK).

- SET.LRCL - Pick up buffer pointers and length of data to transfer.

7.11.6 Data Transfer for Indexed Files (INDEXED)

The purpose of the routine INDEXED is to intercept all SVC 1 requests to an indexed file. If the call is a command function, INDEXED branches to CMD.IN. Otherwise, INDEXED determines the type of I/O request, checks the device busy flag (if a physical I/O is to be done), flushes buffers, if necessary, by a call to RESET.IN, repositions the file, if necessary, and sets the initial value for FCB.CINX (current index block offset) and FCB.COFF (current data block offset) by a call to CALC.NBK. If the call is a read, INDEXED transfers control to INX.GETL. If the call is a write, control is transferred to INX.PUTL.

7.11.7 Command Requests for Indexed Files (CMD.IN)

The routine CMD.IN receives control from INDEXED whenever a command request is directed to an indexed file. CMD.IN contains six executors, to perform the six permissable indexed file command functions. The functions are:

- REWIND (CIN.REW) - Set the files current logical record number (FCB.CLRL) to zero.

- BACKSPACE RECORD (CIN.BSR) - If the file is currently positioned at the beginning, return EOF status, otherwise, decrement FCB.CLRL by one.

- FORWARD SPACE RECORD (CIN.FSR) - Return EOF if file is currently positioned at the end, otherwise, increment FCB.CLRL by one.

- WRITE FILE MARK (CIN.EXIT) - Write file mark is not supported on indexed files.

- FORWARD SPACE FILE (CIN.FSF) - Update FCB.CLRL to contain the value of FCB.CSEC (the number of logical records) to position just beyond the last logical record.

- BACK SPACE FILE (CIN.REW) - Identical to rewind.

Upon successful completion, all six executors return to FMGR.IX1.

CHAPTER 8
EXECUTIVE TASKS AND SYSTEM EXTENSIONS

8.1 INTRODUCTION

There are several ways of extending or modifying the capabilities of OS/16 MT2. This chapter discusses the features designed into OS/16 to facilitate such extensions. The user may wish to incorporate the modification directly into the system by modifying one or more system modules or by adding a system module. For example, the user may support a non-standard peripheral device by writing a driver. Alternatively, the user may wish to support infrequently used extensions to the system by writing an executive task (E-Task) which may be loaded and executed on demand.

8.2 EXECUTIVE TASKS

An executive task (E-Task) is written as a user task and executed in the ET state by specifying OPTION ET when TETing or when loading the task. E-Tasks execute in a hardware and software privileged mode. Privileged machine instructions are allowed and additional privileges are given as follows:

- All addresses are valid in SVC calls
- A disc device (as distinct from a file on the disc) may be assigned
- SVC 2 code 0 (journal entry) is valid (see Figure 8-1)
- REPROTECT (SVC 7) for a key of X'FF' and to devices is valid
- RENAME (SVC 7) to devices is valid
- ASSIGN (SVC 7) to files with any protection keys is valid

As a direct result of these added capabilities, E-Tasks must be designed and coded with extreme caution to prevent crashing the system. The operating system assumes E-Tasks are fully debugged tasks.

CAL Code

```
          ALIGN ADC
PARBLK    DB    0,0
          DC    H'JCODE'
          DAS   4
```

Format

0(0) 00	1(1) 00(00)
2(2) Journal Code	
4(4) RC	
6(6) RD	
8(8) RE	
10(A) RF	

Figure 8-1 SVC 2 Code 0 Parameter Block

Access to system tables and control information is provided through the System Pointer (SPT). The address of the SPT is contained in the half-word at location X'62' in low memory.

E-Tasks may use all SVCs. An example of a function which requires an E-Task is a disc utility. Also, the OS/16 MT2 Command Processor executes as an E-Task.

8.3 SYSTEM EXTENSIONS

OS/16 MT2 may be extended or modified by incorporating changes into the source of one or more system modules or by adding a system module. A system module may be included after the executive module and before the command processor module. System data structures should be referenced using the STRUC which defines the data structure. These field definitions should be used in all instructions referencing the structure. The STRUC is copied in at assembly time from the parameter and control block module (PCB).

Chapter 4, describing the system overlay scheme, outlines coding conventions that must be followed when modifying the operating system. In

particular, note that absolutely located code must not be inserted because TET/16 does not handle it in ESTABLISH OS mode.

8.4 PATCHING

In making modifications to OS/16 MT2, debugging usually involves making patches to new or existing code to avoid reassembling every time a bug is found. It is recommended that a non-overlayed system be generated after making modifications to OS/16. In order to insert a patch in OS/16 MT2:

1. Locate the address UBOT in the map of the system.
2. Use the MODIFY command to increase the value of UBOT by an amount sufficient to contain the patch.
3. Use the MODIFY command to insert the patch starting at the old value of UBOT.
4. Use the MODIFY command or the console panel to insert a branch to the patch area.
5. Restart the system at location X'60' in order to set up the partition table.

APPENDIX 1
JOURNAL AND CRASH CODES

A1.1 CRASH CODES

After a system crash, register 5 contains a pointer to the system journal, register 6 contains a pointer to the most recent entry in the journal, and registers E and F contain the PSW at the time the SINT was executed. The crash code is stored at SPT.CRSH.

Table A1-1 is a list of crash codes, their meanings, and in some cases additional information concerning the cause of the crash.

TABLE A1-1 CRASH CODES

CRASH CODE	DESCRIPTION
2	Invalid queue entry on the command processor's queue. Entry CRSHSV+X'1C' = invalid item.
7	Invalid VMT during MARK processing. Entry CRSHSV+X'14' = address of DMT entry.
100	Arithmetic fault in system code. X'48' = PSW at time of fault.
101	Memory protect fault within system code. Entry CRSHSV+X'1C' = PSW at time of fault.
102	Illegal instruction in system code (with RF pointing within IIMPH routine) or internal interrupt with no current task (with RF pointing within EXECSUB routine). For illegal instruction, entry IIIPSW=PSW at time of interrupt. For internal interrupt, SAVCC can be used to determine the PSW location (see Table 5-4, Chapter 5). In both cases, GRSPTR points to the registers at the time of interrupt.
121	Error in marking on OS volume (in system initialization). e.g. device not configured in system (device number accessed from X'7A' as set by the operator) or unable to read VD due to an I/O error.

TABLE A1-1 CRASH CODES (Continued)

CRASH CODE	DESCRIPTION
132	Illegal SVC in system code. Entry CRSHSV+X'1C' = PSW pointing after SVC instruction
142	Illegal address in SVC in system code. Entry CRSHSV+X'1C' = PSW pointing after SVC instruction.
152	Memory parity error in system code. Entry CRSHSV+X'1C' = PSW at time of interrupt.
153	Attempt to recover from power fail on system with no power fail recovery logic (DELETE PFREC).
162	Illegal interrupt on device zero. RE-RF = PSW at time of interrupt.
321	Attempt to release a free sector. Entry CRSHSV+X'1C' = fullword sector address.
330	Attempt to manipulate allocation for an invalid sector address. Entry CRSHSV+X'C' = fullword sector address.

A1-2 JOURNAL CODES

Table A1-2 contains System Journal Codes.

TABLE A1-2 JOURNAL CODES

CODE	DESCRIPTION AND REGISTER CONTENTS
10	Item removed from system queue (IOTERM) 12 - TCB index 13 - Item removed (address of DCB+1) 14 - NA 15 - NA
11	Task scheduled (SCHED) 12 - NA 13 - NA 14 - Status portion of PSW to be loaded 15 - Location counter of PSW to be loaded
13	Illegal Instruction Interrupt (EXECSUB) 12 - NA 13 - NA 14 - Status portion of PSW at time of interrupt 15 - Location counter of PSW at time of interrupt

TABLE A1-2 JOURNAL CODES (Continued)

CODE	DESCRIPTION AND REGISTER CONTENTS
14	Floating Point Fault Interrupt (EXECSUB) 12 - NA 13 - NA 14 - Status portion of PSW at time of interrupt 15 - Location counter of PSW at time of interrupt
15	System Queue Service Interrupt (EXECSUB) 12 - NA 13 - NA 14 - Status portion of PSW at time of interrupt 15 - Location counter of PSW at time of interrupt
16	Fixed Point Divide Fault Interrupt (EXECSUB) 12 - NA 13 - NA 14 - Status portion of PSW at time of interrupt 15 - Location counter of PSW at time of interrupt
17	Memory Protect Fault Interrupt (EXECSUB) 12 - NA 13 - NA 14 - Status portion of PSW at time of interrupt 15 - Location counter of PSW at time of interrupt
20	Item added to task queue (ADTSKQ) 12 - Return address 13 - Address of TCB 14 - Address of task queue 15 - Parameter to be added to queue
6x	Execution of SVC x (EXECSUB) 12 - First halfword of SVC parameter block 13 - Address of parameter block 14 - SVC old PSW status 15 - SVC old PSW location (updated)
80-FF	User Journal Code (SVC 2 code 0)

APPENDIX 2
DATA STRUCTURES

A2.1 INTRODUCTION

This appendix presents the formats of system control blocks and table entries. Each field is identified by its name and a descriptive title. All control blocks and table entries are referenced in OS/16 MT2 by copying the CAL STRUC of the same name from the OS/16 MT2 Parameters and control block module (PCB). The full field identifier is of the form:

BBB.FFFF

where BBB is the control block name and FFFF is the field name. Most fields are self explanatory. Those which are not are explained following the figure for that control block. Offsets are given in the form:

DD(HH)

where DD is the offset in decimal and HH is the offset in hexadecimal.

A2.2 SYSTEM DATA STRUCTURE RELATIONSHIPS

Figure A2-1 shows the data structure relationships for the system.

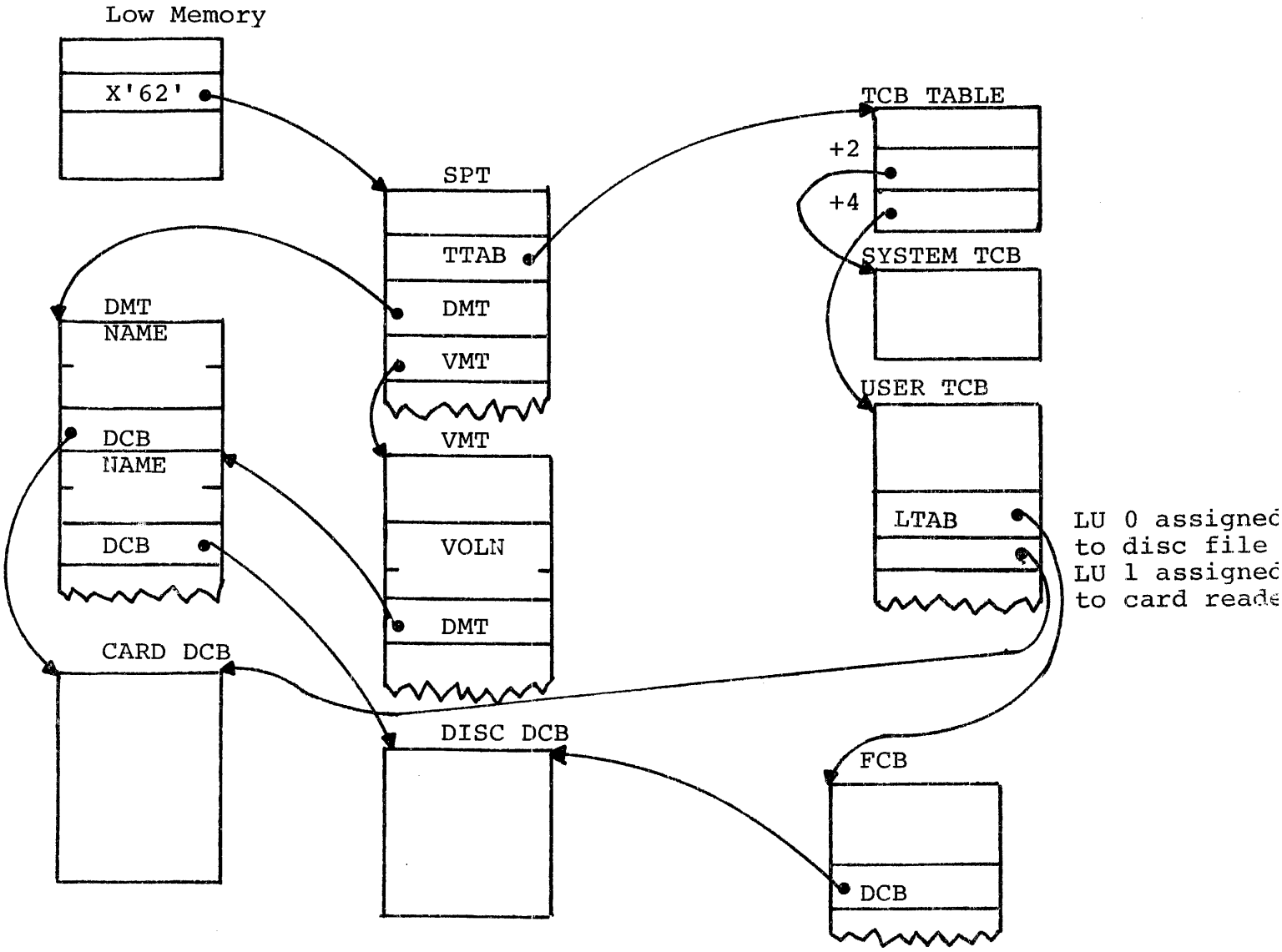


Figure A2-1 System Data Structure Relationships

A2.3 SYSTEM POINTER TABLE (SPT)

Figure A2-2 is a System Pointer Table.

0(0)	INIT		
	branch to SYSINIT		
12(C)		CRSH	system crash code
14(E)		CE	command processor initialization entry
16(10)		UBOT	first byte above OS/16
18(12)		FBOT	lower bound of FCBs
20(14)		MTOP	first byte above top of memory
22(16)		OSID	
	System ID = OS16MT2r		r=release level
30(1E)		TTAB	address of TCB table
32(20)	NTCB number of TCBs	33(21)	CTCB current TCB index
34(22)		DMT	address of DMT
36(24)		VMT	address of VMT
38(26)		JRNL	address of system journal
40(28)		FLBA	
	starting sector number of OS overlay file		
44(2C)		OVFD	
	name of OS overlay file (VOLN/FILENAME/EXT)		
60(3C)		SDCB	pointer to DCB of OS overlay device

Figure A2-2 System Pointer Table (SPT)

A2.4 TASK CONTROL BLOCK TABLE (TCBTAB)

TCBTAB is pointed to by SPT.TTAB.

There is no STRUC for the TCB TAB.

Figure A2-3 is a Task Control Block Table.

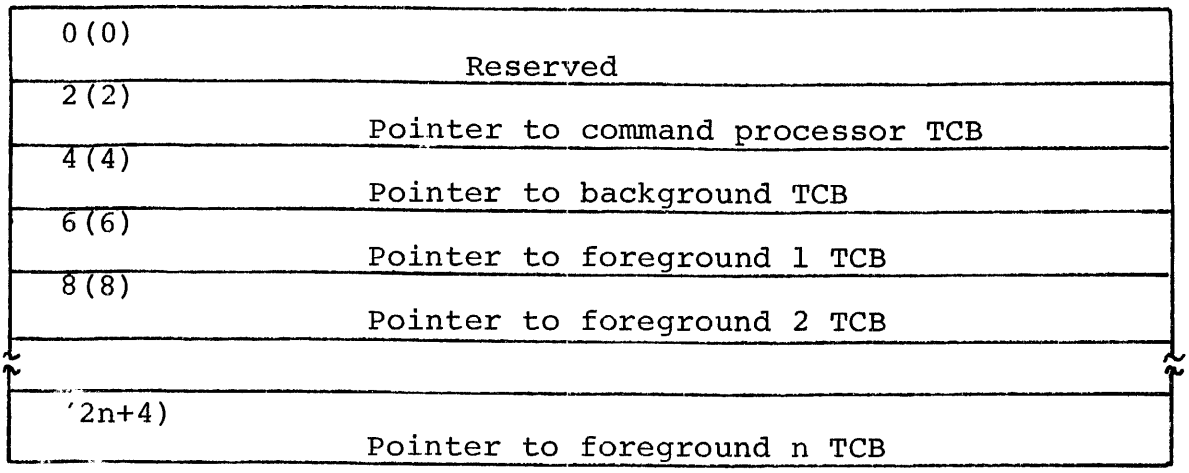


Figure A2-3 Task Control Block Table (TCBTAB)

A2.5 TASK CONTROL BLOCK (TCB)

Figure A2-4 illustrates a Task Control Block

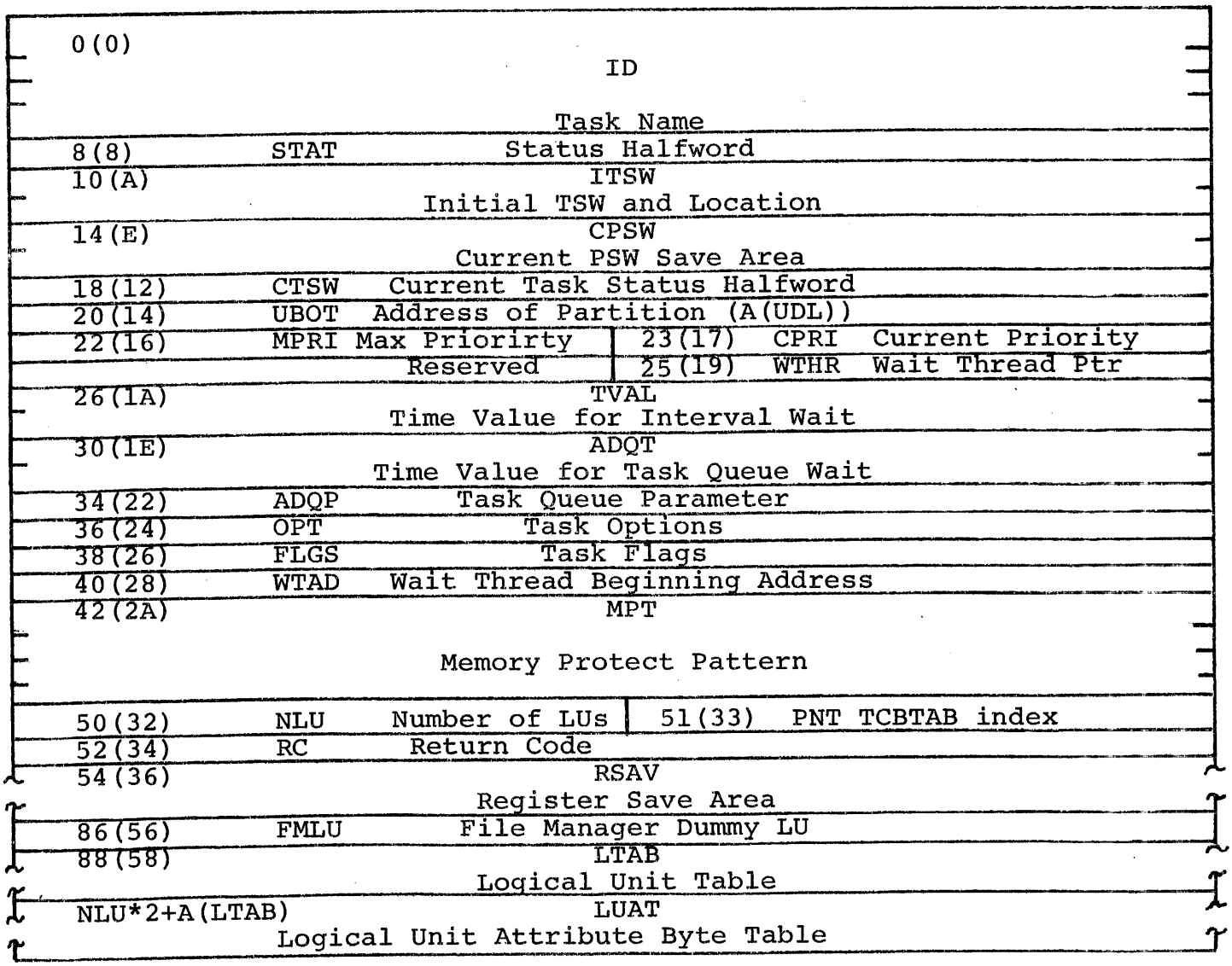


Figure A2-4 Task Control Block (TCB)

A2.5.1 Status (TCB.STAT)

Mask	Flag Name	Meaning if Set
8000	TSTT.DMM	Dormant. Task loaded but not started or task has gone to EOT.
4000	TSTT.IOM	I/O wait.
1000	TSTT.RWM	Roll wait. Task is being rolled out or task is being rolled in.
0400	TSTT.CWM	Console wait. Task is paused
0200	TSTT.TIM	Time wait. Task is in interval/time wait.
0100	TSTT.TRM	Trap wait.
0080	TSTT.S2M	SVC 2 wait.
0040	TSTT.S5M	SVC 5 or 6 wait.
0020	TSTT.S7M	SVC 7 wait.
0008	TSTT.DWM	DCB wait. Task is waiting for a disc DCB.

A2.5.2 Options (TCB.OPT)

Mask	Flag Name	Meaning if Set
8000	TOPT.ETM	Task is an E-Task.
4000	TOPT.ACM	Continue on arithmetic fault.
2000	TOPT.FPM	Task uses single precision floating point.
1000	TOPT.MRM	Task is memory resident.
0800	TOPT.COM	Task uses compatible SVCs.
0400	TOPT.BGM	Task is the background task.
0200	TOPT.S6M	SVC 6 from the background will be ignored.
0100	TOPT.ROM	Task is rollable.
0040	TOPT.DFM	Task uses double precision floating point.

A2.5.3

Mask	Flag Name	Meaning if Set
0080	TFLG.PPM	Task to be put into console wait.
0040	TFLG.CPM	Task is being cancelled or is going to EOT.
0020	TFLG.SLM	Task is executing in ET state in the system.
0010	TFLG.BIM	Task is executing in ET state in the BISYNC driver.
0008	TFLG.PRM	Task was in system when power restore took place.
0004	TFLG.ROM	Task is to be rolled out.
0002	TFLG.TOD	Task is in time of day wait.
0001	TFLG.TPM	A TSW swap is to take place.

A2.6 DEVICE MNEMONIC TABLE (DMT)

The DMT consists of one entry for each device configured in the system. The table is terminated by a halfword of zero. The DMT is pointed to by SPT.DMT. The DMT structure is called 'DMT.' Figure A2-5 is a Device Mnemonic table entry.

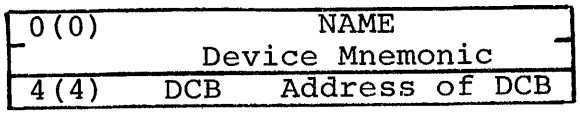


Figure A2-5 Device Mnemonic Table (DMT)

A2.7 DEVICE CONTROL BLOCK (DCB)

The DCB is used to identify characteristics of each device configured in the system and to serve as a work space for drivers during an I/O request. DCBs are pointed to by the DMT entry for the device represented. DCBs are included in the system at source SYSGEN time. Figure A2-6 is a Device Control Block.

-22(-16)		RECL	Record Length (0 = variable)
-20(-14)	WCNT	Write Count	-19(-13) RCNT Read Count
-18(-12)	DCOD	Device Count	-17(-11) DN Device Number
-16(-10)		ATRB	Attributes of Device
-14(-E)	WKEY	Write Key	-13(-D) RKEY Read Key
-12(-C)		BUSY	Address of Busy Flag
-10(-A)		TERM	Address of Driver Termination Routine
-8(-8)		ALOC	Address of Driver Abort Routine
-6(-6)		TOUT	Time Out Count
-4(-4)		FLGS	Flags Halfword
-2(-2)		INIT	Address of Driver Initialization Routine
0(0)			OPSW
			Old PSW Save Area
4(4)		NPSS	New PSW Status (ISSTAT)
6(6)			ENTR
			Interrupt Service Entry Point
16(10)			LEAV
			Interrupt Service Exit Location
20(14)			NOPI
			NOP Interrupt Service Entry Point
28(1C)	RSAV	R2	TCB Pointer | 29(ID) IOW I/O Wait Pointer
30(1E)		R3	Address of parameter block
32(20)		R4	Function Code/LU
34(22)		R5	
36(24)		R6	Device Number
38(26)		R7	Status
40(28)		R8	
42(2A)		R9	
44(2C)		RA	Length of data transfer
46(2E)		RB	
48(30)		RC	
50(32)		RD	Device Number
52(34)		RE	Address of DCB
54(36)		RF ISR	Interrupt Service Routine Ptr
56(38)		RES	Reserved for busy flag

Figure A2-6 Device Control Block (DCB)

A2.7.1 Flags (DCB.FLGS)

Mask	Flag Name	Meaning if Set
8000	DFLG.BLM	Bulk device (disc)
4000	DFLG.LNM	Device is on line
2000	DFLG.BBM	BISYNC device is busy
1000	DFLG.MPM	Valid bit map record is in memory
0800	DFLG.PFM	Disc driver should check record for pseudo file mark
0400	DFLG.BMM	Bit map record in memory has been modified
0200	DFLG.CNM	Device is the console device
0100	DFLG.BIM	Device is a BISYNC device
0080	DFLG.S6M	Device is connectable
0040	DFLG.WPM	Device is write protected
0020	DFLG.HIM	Halt I/O in progress on device
0010	DFLG.S1M	Routine was called from SVC 1
0008	DFLG.S0M	Bit map was searched from start
0004	DFLG.IUM	DCB is in use
0001	DFLG.SDM	SELCH device

A2.7.2 Attributes (DCB.ATRB)

Mask	Flag Name	Meaning if Set
8000	DATR.INM	Interactive device
4000	DATR.RDM	Supports read
2000	DATR.WRM	Supports write
1000	DATR.BIM	Supports binary formatted records
0800	DATR.WTM	Supports wait I/O
0400	DATR.RNM	Supports random requests
0200	DATR.UMP	Supports unconditional proceed
0100	DATR.IMM	Supports image
0080	DATR.HIM	Supports halt I/O
0040	DATR.RWM	Supports rewind
0020	DATR.DRM	Supports backspace record
0010	DATR.FRM	Supports forward space record
0008	DATR.WFM	Supports write file mark
0004	DATR.FFM	Supports forward space file mark
0002	DATR.BFM	Supports backspace file mark

A2.8 VOLUME MNEMONIC TABLE (VMT)

There is one entry in the VMT for each disc device configured in the system. When a disc is marked on line, the volume name is read from the VD and placed in the VMT entry corresponding to the disc being marked. The VMT is terminated with a halfword of zero. The VMT structure is called 'VMT.' Figure A2-7 is a Volume Mnemonic Table entry.

0(0)		VOLN Volume Name
4(4)	DMT	Address of Corresponding DMT entry

Figure A2-7 Volume Mnemonic Table (VMT)

A2.9 FILE CONTROL BLOCK (FCB)

Figure A2-8 illustrates a File Control Block.

0(0)		VMT	Address of VMT Entry		
2(2)	WCNT	Write Count		3(3) RCNT	Read Count
4(4)		ATRB	Attributes of File		
6(6)		LRCL	Logical Record Length		
8(8)	OFF	Directory Offset		9(9) BKSZ	File Blocksize
10(A)		DIR Address of Directory Block			
14(E)	FLGS	Flags		15(F) DCOD	Device Code
16(10)		NAME Filename			
24(18)		EXT	File Extension		
				27(1B) VERS	Reserved
28(1C)		DCB	Address of DCB		
30(1E)		FLBA First Logical Block Address			
34(22)		LLBA Last Logical Block Address			
38(26)		CSEC Current Sector/Number of Logical Records			
42(2A)	LU	Logical Unit		43(2B) TPNT	TCB Pointer

Figure A2-8 File Control Block (FCB)

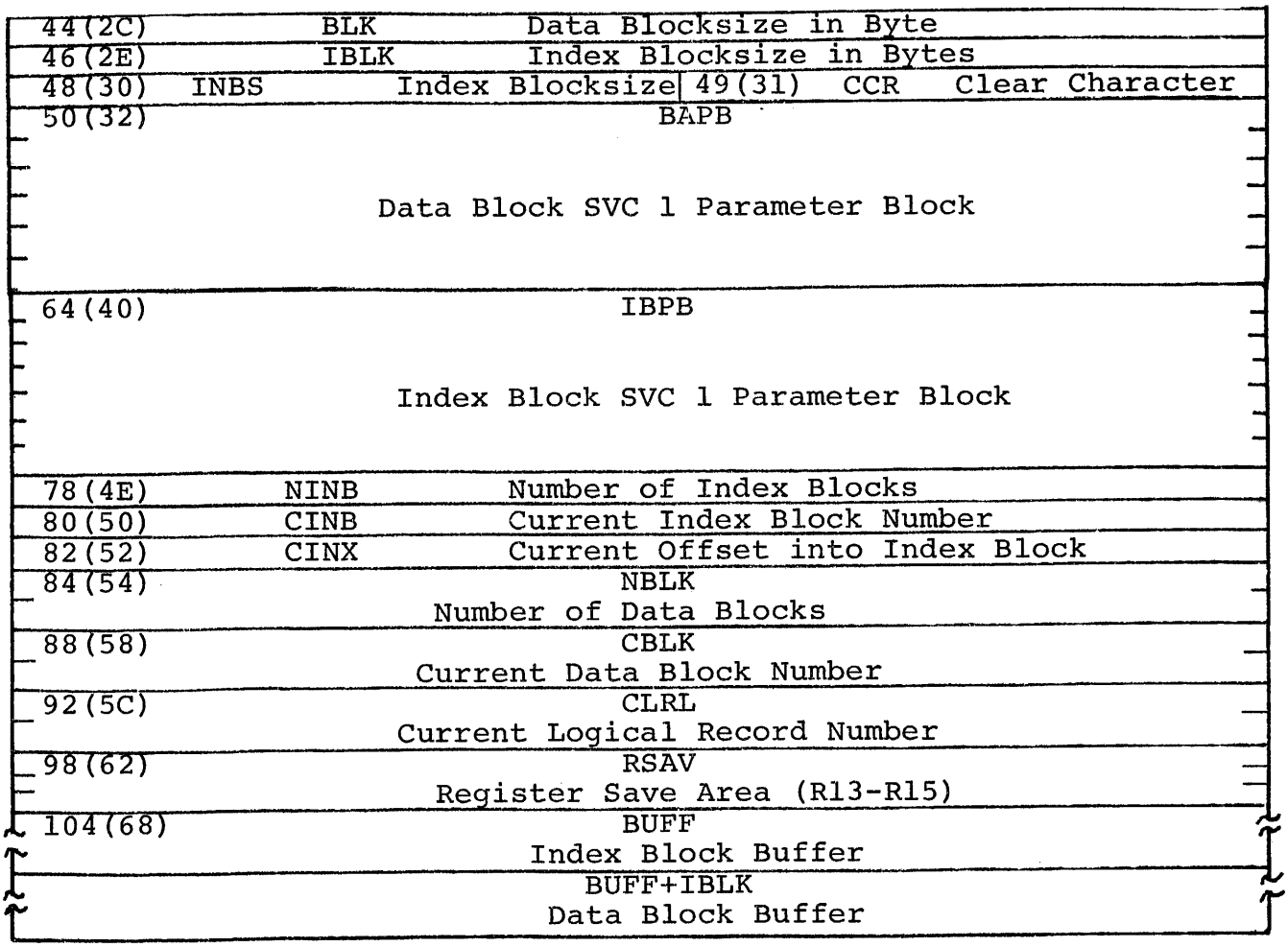


Figure A2-8 File Control Block (FCB) (Continued)

(Additional for Indexed Files Only)

A2.9.1 Flags (FLGS)

Mask	Flag Name	Meaning if Set
80	FFLG.BAM	Buffered access method
40	FFLG.IUM	FCB is in use; SVC 1 request to file is being processed.
20	FFLG.OPM	Write request is being processed.
08	FFLG.BMM	Data block buffer has been modified.
04	FFLG.MOM	Extending the file; increasing the number of logical records.
02	FFLG.SBM	Disc and SELCH busy flags were set.
01	FFLG.XMM	Index block buffer has been modified.

A2.9.2 Device Code (DCOD)

Value	Flag Name	Meaning
00	FDCD.CO	Contiguous file
02	FDCD.IN	Indexed file

A2.10 VOLUME DESCRIPTOR (VD)

The volume descriptor is written onto sector 0 of the disc by the INITIALIZE command. VD.OSP, and VD.OSS fields are not used by OS/16. Figure A2-9 illustrates a Volume Descriptor.

0(0)	VOL Volume Name
4(4)	ATRB Volume Attributes
8(8)	FDP First Directory Block Pointer
12(C)	OSP Pointer to OS Image
16(10)	OSS Size of OS Image
20(14)	MAP Pointer to Bit Map

Figure A2-9 Volume Descriptor (VD)

Attributes (ATRB)

Mask	Flag Name	Meaning if Set
8000	VATR.ONM	Disc is marked on-line

A2.11 DIRECTORY ENTRY (DIR)

Each directory block contains five directory entries. DIR.VERS, DIR.DATE, and DIR.LUSE are unused in OS/16MT2. Figure A2-10 illustrates Directory Entry.

0(0)	FNM File Name		
8(8)	EXT Extension	11(B)	VERS Version
12(C)	FLBA First Logical Block Address		
16(10)	LLBA Last Logical Block Address		
20(14)	WKEY Write Key	21(15)	RKEY Read Key
22(16)	LRCL Logical Record Length		

Figure A2-10 Directory Entry (DIR)

24(18)		DATE Creation Date/Time	
28(1C)		LUSE Last Used Date/Time	
32(20)	WCNT	Write Count	
34(22)	RCNT	Read Count	
36(24)	ATRB	File Attributes	37(25) BKSZ Data Blocksize
38(26)	INBS	Index Blocksize	39(27) Reserved
40(28)		CSEC Current Sector (Contiguous)/Number of Logical Records (Indexed)	
44(2C)		Reserved	

Figure A2-10 Directory Entry (DIR) (Continued)

File Attributes (DIR.ATRB)

Value	Meaning
00	Entry is free; contains no valid information
10	Contiguous file
50	Indexed file

A2.12 LOADER INFORMATION BLOCK (LIB)

Every load module produced by TET/16 consists of a loader information block followed by the image of the task, library or task common that was established. The LIB passes to the resident image loader all information about the module which must be known before the load can proceed. Figure A2-11 illustrates a Loader Information Block.

0(0)	OPT	Task Options	1(1) FLGS	Task Flags
2(2)	MPRI	Maximum Priority	3(3) IPRI	Initial Priority
4(4)		UBOT	Address of Partition	
6(6)		UTOP	Top of program	
8(8)		CKSM	Checksum	
10(A)		CTOP	Top of Partition	
12(C)	ID	SRL	Address of Reentrant Library	
14(E)	ID	overlay name	SCM Address of Task Common	
16(10)	ID		reserved	
18(12)		OBOT	Overlay bottom	

Figure A2-11 Loader Information Block

A2.12.1 Options (OPT)

Mask	Flag Name	Meaning if Set
8000	LOPT.ETM	Task is an E-Task
4000	LOPT.ACM	Continue on arithmetic fault
2000	LOPT.FPM	Task uses single precision floating point
1000	LOPT.MRM	Task is memory resident
0800	LOPT.COM	Task uses compatible SVCs
0200	LOPT.S6M	SVC 6 from the background to be ignored
0100	LOPT.ROM	Task is rollable
0040	LOPT.DFM	Task uses double precision floating point

A2.12.2 Flags (FLGS)

Mask	Flag Name	Meaning if Set
80	LFLG.OVM	Module is an overlay

A2.12.3 Checksum (CKSM)

A checksum is formed from the entire load module image (excluding the LIB data).

The checksum halfword is initialized to -1 and dynamically updated in a loop, forming the exclusive OR with each halfword of data in the module.

This is checked by the loader against the image data after it has been loaded into memory. If they do not agree, then data has been lost or corrupted.

PUBLICATION COMMENT FORM

Please use this postage-paid form to make any comments, suggestions, criticisms, etc. concerning this publication.

From _____ Date _____

Title _____ Publication Title _____

Company _____ Publication Number _____

Address _____

_____

_____

FOLD FOLD

Check the appropriate item.

☐ Error Page No. _____ Drawing No. _____

☐ Addition Page No. _____ Drawing No. _____

☐ Other Page No. _____ Drawing No. _____

Explanation:

FOLD FOLD

CUT ALONG LINE

Fold and Staple

No postage necessary if mailed in U.S.A.

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT No. 22
OCEANPORT, N. J.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY:

INTERDATA®

Subsidiary of PERKIN-ELMER
Oceanport, New Jersey 07757, U.S.A.

TECH PUBLICATIONS DEPT. MS 322

FOLD FOLD

STAPLE STAPLE