Table of Contents

Table of Contents (Continued)

Table of Contents (Continued)

APPENDICES

ILLUSTRATIONS

Table of Contents (Continued)

TABLES

CHAPTER 1
SYSTEM OVERVIEW

INTRODUCTION

OS/16 MT2 is a multi-tasking Operating System designed for INTERDATA 16-Bit Architecture Processors. The user should be familiar with the information in the following documents:

16-Bit Processor User's Manual, Publication Number 29-509

This manual is intended as a programming reference manual for OS/16 MT2. For information on operating OS/16 MT2 refer to·

OS/16 MT2 Operators Manual, Publication Number 29-430

Other manuals related to OS/16 MT2 are:

OS/16 MT2 System Planning and Configuration Guide, Publication Number 29-431
OS/16 MT2 Pocket Guide, Publication Number 29-433
OS/16 MT2 Program Logic Manual, Publication Number 29-434

A complete bibliography and subject index are contained in Appendix 9 of the *OS/16 MT2 System Planning and Configuration Guide*

The OS/16 MT2 Multi-Tasking Operating System provides an efficient and powerful means for using the resources of an INTERDATA 16-Bit Processor and its environment. Through easy to use services and efficient resource management, the user is free to concentrate on problem solving rather than on system management OS/16 MT2 provides:

System control services through the Command Processor

Program control services through the Task Management Facilities of the Executive.

Memory management through the Executive.

Direct Access and Non-direct Access data management services through the File Manager and Device Drivers.

Figure 1-1 illustrates the components of OS/16 MT2 and their relationships.

System control is supported both from the system console and from user specified files or devices. The OS/16 MT2 Command language may be extended through the facilities of the Command Substitution System (CSS) to allow invocation of complex control sequences with a single command.

Program control support includes background and foreground facilities so that program preparation can proceed concurrently with real-time system operation. Intertask communication, task handled traps, single tasking, multi-tasking and parallel tasking support is provided.

Memory management facilities include multiple task common areas and resident library partitions, and multiple overlay area capabilities for user tasks. Storage management services are provided by the Executive for orderly memory management by user tasks. Foreground tasks are protected from undebugged background tasks. User space is divided into partitions established at system generation time. Partition sizes can be changed dynamically by the system operator

The OS/16 MT2 File Manager and Device Drivers provide a powerful set of data management services for both Direct Access and non-Direct Access devices. Two types of Direct Access file organization are supported: Contiguous and Indexed. Both static and dynamic protection is provided for I/O devices and Direct Access files.
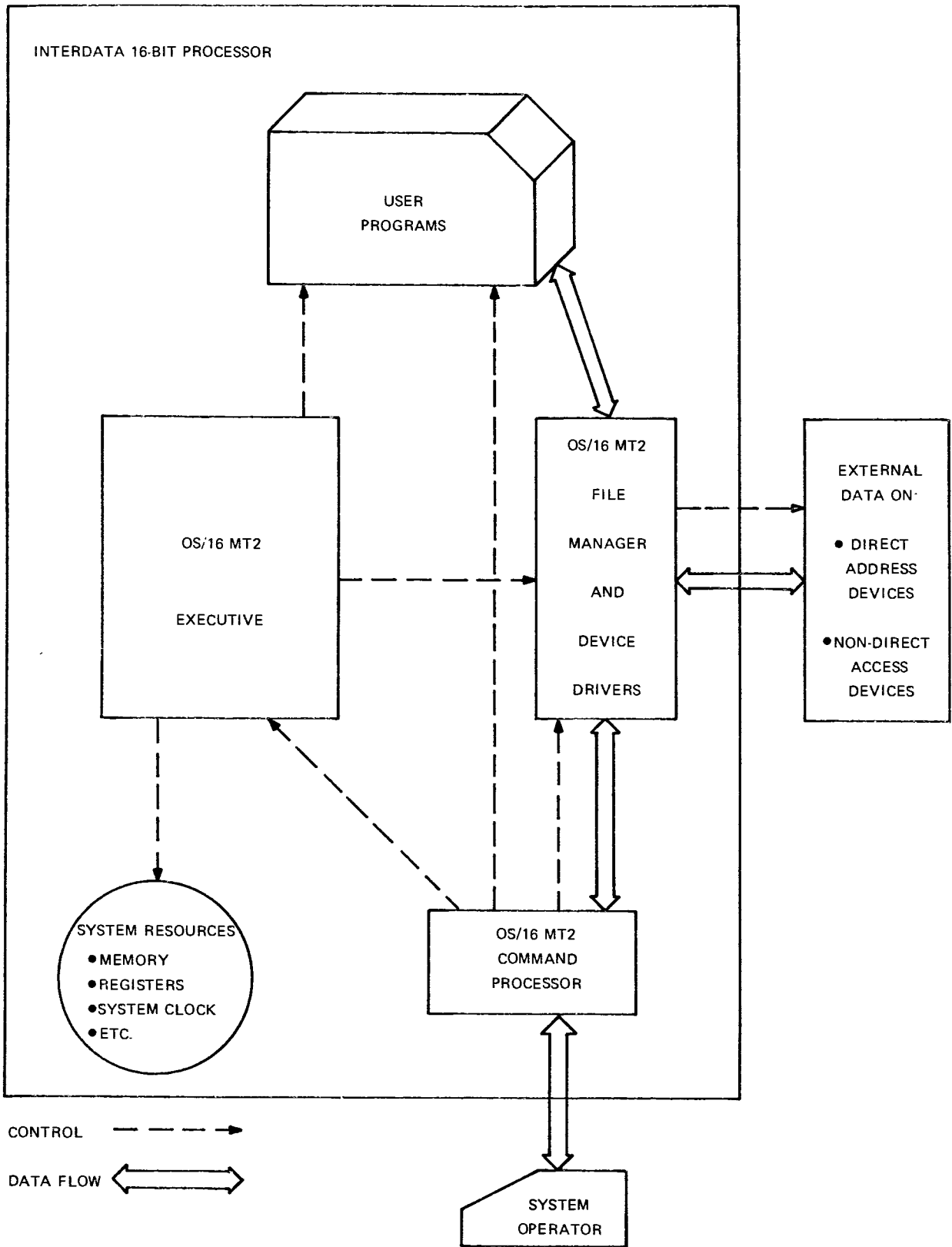
Figure 1-1. OS/16 MT2 System Overview

The OS/16 MT2 user can access the facilities of the system through either of two interfaces.

1. Operator interface; provided by the command language of the OS/16 MT2 Command Processor.

2. Program interface: provided by Supervisor Call (SVC) instructions serviced by the OS/16 MT2 Executive and File Manager.

In order to provide powerful facilities in the most efficient manner, an OS/16 MT2 System can be tailored to a specific hardware and system environment through the OS/16 MT2 system generation (SYSGEN) procedure. In addition, non-critical portions of the Operating System may be maintained on a Disc and brought into memory only when necessary, greatly reducing the memory requirements of the system

OS/16 MT2 can function as a subset of OS/32 MT for applications programming. Indexed and Contiguous Disc files are compatible between OS/16 MT2 and OS/32 MT.

MANUAL ORGANIZATION

This manual is intended for use as a programming reference manual. Chapter 1 describes the system in general terms. Chapter 2 provides information on task control at the program level. Chapter 3 describes the Supervisor Call (SVC) instructions supported by OS/16 MT2. Chapters 4, 5, 6, and 7 provide introductory guides to the use of SVC 2 facilities, task handled traps, Direct Access files, and user task overlays. Chapter 8 provides detailed device dependent information for I/O programming.

SYSTEM DESCRIPTION

Tasks

Program control services are provided for controlling a single task or a complex system of tasks. A task may consist of a single program, or it may include a main program with a number of subroutines and overlays. Tasks may be permanently resident in memory, or they may be loaded as required. Tasks are referred to by a Task Identifier (TASKID) which is associated with the task at load time. The number of tasks allowed in memory at one time is limited only by the number of partitions established at system generation time.

The following sections describe two pairs of task categorizations.

Foreground/Background Tasks

Tasks resident in a foreground memory partition are referred to as foreground tasks while a task resident in the background partition is referred to as the background task. Foreground tasks have the full range of OS/16 MT2 services available while a background task has the following restrictions·

Intertask communication facilities are not supported.
Task priority is under console operator control.
A background task may not communicate with the foreground tasks through Task Common

This prevents a background task from interfering with the foreground system, thus providing a safe environment for task debugging.

Tasks may be established as tasks by processing the main program and any subroutines or overlays with the OS/16 Task Establisher Task (TET/16). The output of TET/16 may then be loaded by a foreground task or by an operator command. TET/16 output consists of a record which contains the Loader Information Block (LIB) followed by records which contain a memory image of the established task. The LIB contains information needed by the resident image loader to ensure proper loading of the task.

Alternatively, a task may be loaded directly in object format (output of CAL − the INTERDATA Common Assembler) by the background or the foreground object load operator commands, if it consists of a single program, or after processing by the OS/16 Library Loader, if it consists of multiple programs and subroutines.

User/Executive Tasks

At task establishment time, a task may be designated as a User task or an Executive task. User tasks (U-tasks) run in a protected mode while Executive tasks (E-tasks) have the full range of 16-bit architecture capabilities available. E-tasks capabilities are designed to provide an orderly and well defined means for adding system extensions.

Previously Written 16-bit programs

Programs written for previous operating systems (BOSS, DOS, or RTOS) must run with the compatibility option, due to differences in the SVC 1 and SVC 5 parameter blocks See the *OS/16 MT2 Operators Manual.*

Program Addresses and Logical Segments

The process of establishing a user task via TET/16 produces a memory image load module of the task This means the task may be loaded by simply reading it into the appropriate memory partition. At assembly time, whether the task consists of assembly language programs or FORTRAN output, the program references data and instructions using program addresses which are usually relative to the first location of the task. At task establish time, these addresses are converted to addresses within the partition that the task is being established for, or to addresses within the specified task common or library partition (see Chapter 5 of the *OS/16 MT2 Operator's Manual* for task establishment procedures). Thus an established task can contain references to several disjoint ranges of contiguous addresses: addresses in the program's partition, those in a task common partition, and those in a library partition Each range of contiguous addresses is termed a *Logical Segment* The three possible logical segment types are

> Program Segment
> Library Segment
> Task Common Segment

Status

A task in memory may be in any of five states These are:

> Current
> Ready
> Wait
> Paused
> 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 execute 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 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.

Protection

User tasks run in a protected mode. This means that they cannot use any privileged instruction. Privileged instructions include all I/O instructions and any instruction that changes the state of the Processor, such as LPSW, EPSR.

In order to request I/O functions or any Processor state change, user tasks must use a Supervisor Call (SVC) instruction.

Command Processor Services

In OS/16 MT2, the console operator interface is provided by a task called the Command Processor. The Command Processor is an E-task which runs as the highest priority task in the system. All commands are interpreted and executed by the Command Processor which also performs all I/O requests to the console device. The *OS/16 MT2 Operators Manual* describes in detail the commands and procedures related to the Command Processor. The Command Processor task is included as a part of OS/16 MT2 and cannot be cancelled. When the Command Processor is the only task in the system (if there is a background task it must be Dormant), the system is said to be quiescent.

The Command Processor includes·

       Support for controlling the user memory partitions

       Command Substitution System (CSS) support which allows the OS/16 MT2 operator command language to be extended by user defined command sequences.

       Support for controlling I/O devices configured in the system.

       The object loaders which allow tasks to be loaded into memory.

The Command Processor module can be deleted from the system at system generation. Planning and programming for such systems is fully described in Chapter 2 of *OS/16 MT2 System Planning and Configuration Guide*

Memory Management

Memory in an OS/16 MT2 system is separated into two classes:

       User space
       System space

System space is discussed later. User space is divided into partitions. The maximum number of partitions is established at system generation time The console operator may vary the sizes of the partitions when there is no task in an affected partition OS/16 MT2 supports four types of partitions

       Foreground
       Background
       Task Common
       Resident Library

Foreground Partitions

Up to 125 foreground partitions may be established at system generation time. Their size is set by operator command. When an established task is loaded by the console operator or by another foreground task, it is loaded into the partition associated with that task at establishment time.

Foreground tasks are permitted to request the loading and execution of other tasks, cancel these tasks or delete them from memory, or to pass parameters to other foreground tasks. In operator commands, a foreground partition is referrenced by the TASK ID of the task loaded in it or by its partition number, if it is vacant.

Background Partition

OS/16 MT2 provides one background partition. Intertask service requests are treated as illegal or ignored (depending on the task option selected) when executed from the background task; therefore the background task may not directly affect the operation of any foreground task. A task in the background partition may not use Task Common nor make use of a resident library.

The size of the background partition is adjusted by the console operator when the start of the first foreground partition is adjusted. The size of the background is set to the memory between the top of the system and the first foreground partition. The background partition is always referenced by its name, .BG.

Task Common

A Task Common partition is a sharable foreground data partition and, as such, is accessible only to foreground tasks. OS/16 MT2 supports multiple Task Common Partitions. When a task is to use a Task Common partition, the user must specify the start address of the desired Task Common partition when establishing the task. At task execution time, the Task Common partition may be vacant, or it may be initialized by loading a task which defines the desired values. Although any task can reference only one Task Common partition, there can be several groups of tasks, each group referencing a different Task Common.

Resident Libraries

A resident library partition is a foreground partition containing sharable, reentrant routines. Multiple resident library partitions are supported by OS/16 MT2, although any task can reference only one resident library partition. All references to routines in a library segment are resolved at task establishment time. A resident library is essentially the same as any foreground task, it is loaded and deleted in the same manner. Background tasks that are established by TET/16 can not use foreground library partitions.

System Space

Certain memory areas in an OS/16 MT2 system are not occupied by any partition. These areas are known as *system space* System space is used in OS/16 MT2 for two purposes: to hold the OS/16 MT2 code itself. and to hold certain tables and system data structures required for proper operation of the system and of the user tasks

The OS itself and all static data structures (those that do not change in size during system execution) are located in the lowest part of physical memory, below the background partition. Dynamic data structures, such as File Control Blocks (FCBs) are located in the highest part of physical memory. These are the only two areas of system space; the remainder of physical memory is devoted to partitions.

Sample Memory Allocation

Figure 1-2 is a map of a hypothetical OS/16 MT2 system. The assumed configuration is a Processor with 64KB of memory. The OS is presumed to occupy 20KB. The partitions are as follows:

Task Common:	1KB
Foreground.	2x4KB, 10KB
Resident Library:	2KB
Background:	16KB

The remaining 7KB is system space.

Memory Address (Hex)

10000 (7KB)	System Space
E400 (1KB)	Foreground Partition 5 (Task Common)
E000 (16KB)	Foreground Partition 4
B800 (4KB)	Foreground Partition 3
A800 (2KB)	Foreground Partition 2 (Resident Library)
A000 (4KB)	Foreground Partition 1
9000 ('6KB)	Background Partition
5000	OS/16 MT2

Figure 1-2. OS/16 MT2 Memory Map Example

Resident Loaders

OS/16 MT2 provides three resident loaders: the memory image loader and the object format background and foreground loaders.

The memory image loader loads overlays, Task Common segments, resident library segments, and either foreground or background tasks. Since this loader loads memory image format, the task to be loaded must have been prepared with OS/16 Task Establisher Task. The image loader may be invoked by a foreground task or by operator command This loader performs all functions as a subroutine of the calling task and it is interruptable but not reentrant. That is, higher priority tasks are not blocked by the loader but only one task can use the loader at a time. If the loader is in use at the time of a loader request, the invoking task is placed in a SVC 6 Wait state.

The object format loaders load tasks in object format (e.g , CAL or OS/16 Library Loader output) and are invoked by operator command.

Roll Out/In

OS/16 MT2 provides one level of automatic roll out for partitions in a Disc-based OS/16 MT2 system. At the time the partitions are established by the operator via the SET PARTITION command, a Contiguous file is allocated for each partition on the specified ROLL volume. On a memory image load request, if a task is to be loaded into a partition which already contains a lower priority task with the rollable option, the lower priority task is rolled out by writing the memory image of the partition to the allocated roll file, along with the necessary control information to restart the task. When the higher priority task is deleted from the system, the rolled out task is rolled in and it resumes execution from the point of interruption A task is rolled out by a load request only if:

> it has the rollable task option set
> it is lower priority than the task being loaded
> there is no task rolled out from the specified partition
> there is no incomplete time interval for the task
> the task is not connected to a Trap Generating Device

If any of the above conditions are not met, the load request is rejected While a task is rolled out, no other task can communicate with it.

Protection

Memory protection is achieved through the use of the Memory Protect Controller (MPC) which is optional in OS/16 MT2. Through the MPC, system space is protected against modification during user task execution and foreground partitions are protected against modification by an undebugged background task. Foreground tasks are allowed to communicate directly with each other. In addition, the Supervisor Call (SVC) handler prevents modification of memory outside a user task's allocation through a request for OS/16 MT2 services.

Task Overlays

To provide more efficient use of memory, OS/16 MT2 supports multiple overlay capabilities for user tasks. At task establishment time, the user may organize a task so that only those portions of the task that are necessary are in memory at any given time. The remainder of the task can be maintained on a device or file until needed. Task establishment facilities, and access to the memory image loader for overlay loading at run time, provide the ability to tailor the memory requirements of a task. Multiple overlay areas may be defined within the task's memory allocation and any number of overlay segments may be loaded into each area as needed, substantially reducing the amount of memory needed to run the task.

Executive Services

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 mediated by the Executive's Supervisor Call (SVC) handler. Other support provided by the Executive includes:

> Software emulation of Single and Double Precision Floating Point, Multiply and Divide, and List Instructions for processors that lack them.

> Support of Line Frequency and Precision Interval Clocks.

> Internal interrupt handling including Power Fail/Restore handling.

> Task scheduling and trap facilities

> External interrupt handling.

> Support via Supervisor Call (SVC) for most Command Processor services

Interrupts and Traps

In OS/16 MT2, as in most operating systems, all interrupts at the Processor level, both external and internal, are handled by the operating system. OS/16 MT2 provides an interrupt facility at the task level known as the *task-handled trap* facility. This facility permits a task to be interrupted from its normal execution sequence for any one of a variety of causes, both hardware and software-generated. This facility also allows one foreground task to pass a parameter to another foreground task via the receiving task's Task Queue, thus permitting the receiving task to determine the software defined reason for the interrupt A task handled trap may occur for the following reasons

    Power Restoration
    Addition of an item to the Task Queue
    Completion of an I/O proceed request
    External Interrupt from a Trap Generating Device
    Completion of a specified time delay
    Message received from another task

Task Status Word (TSW)

Traps and additions to a task's Task Queue are controlled through the task's Task Status Word (TSW), which is the task level analog of the 16-Bit Processor Program Status Word (PSW). This status word is used to enable or disable the various traps, enable or disable additions to the Task Queue, and to save the location counter and condition code of a task at the time of a trap.

Task-Handled Traps

The user task can enable or disable the various traps through manipulation of its Task Status Word; on the other hand, a user may ignore the task trap facility by choosing the default TSW at task establishment time. The following sections briefly discuss the various task-handled trap facilities.

Task Queue Service

OS/16 MT2 supports a Task Queue, which is a circular list in standard INTERDATA list processing format, that may be associated with a task If enabled by its Task Status Word (TSW), the following conditions cause an item to be added to a task's Task Queue:

    Queuing of an item by another task
    Completion of an I/O proceed request
    External Interrupt from a Trap Generating Device such as the Eight-Line Interrupt module.
    Completion of a time interval
    Message received from another task

Power Restoration

A task can request a trap, through its TSW, on completion of the system power restoration sequence, instead of being paused as is the default action. Any I/O in progress has been aborted by the system prior to the trap, except for Direct Access I/O which has been retried. If a task has real-time considerations, power restore may indicate that an appreciable amount of time has elapsed while power was down. A Power Restoration trap may be used to re-establish time intervals or to retreat to a checkpoint for resumption of task execution in an orderly fashion.

Interrupts

The OS/16 MT2 Executive handles the following Processor internal interrupts and OS/16 MT2 generated interrupts in essentially the same way:

    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, default system action is to log a message to the system console and pause the current task. If no task is current (the fault occurs during system code execution), a system crash occurs. If the optional system crash handler is included during system generation, a descriptive crash code is displayed on the display panel and the Processor placed in the Wait state. The user task can override default system action for Arithmetic Faults by either disabling them or causing the system to simply log a message and continue.

Task Priority and Scheduling

OS/16 MT2 recognizes 256 task priorities from a high of 0 to a low of 255. Of these levels, 10 through 249 are available to user tasks while 0 through 9 and 250 through 255 are reserved for system use. Task priority is set initially at task establishment time; it may be altered dynamically by another foreground task or the operator, up to a maximum set at task establishment.

In OS/16 MT2, tasks are scheduled in priority order. Care should be taken in assigning priorities so that tasks which do not relinquish control of the Processor do not prevent lower priority tasks from executing. Tasks on the same priority level are selected for scheduling according to either of two rules selected at system generation time.

1. Tasks residing in lower partition chosen ahead of tasks in higher partition (i.e. Task in partition 1 scheduled ahead of task in partition 2).

2. Round-Robin scheduling The most recently scheduled task on the current highest priority level is selected for scheduling after all other tasks on the same level. In this scheme, a queue is kept of ready tasks ordered by priority. When a task relinquishes control of the system it is placed at the end of its priority queue.

A task relinquishes control of the Processor in any of the following events.

It is Paused by the console operator

It is Cancelled

A higher priority task becomes ready because of an external event.

It Executes a request for an OS/16 MT2 service which places it in a Paused, Wait, or Dormant state.

Data Management Services

In order to provide device independent input/output programming, programs direct all I/O requests to a Logical Unit (LU) number instead of to a specific device or file. The system maintains a Logical Unit Table (LTAB) for each task. LU numbers, which range from zero to a limit set at system generation time (maximum 63), correspond to LTAB entries for the task. The LUs referenced by a task must be assigned to the specific devices or files by the operator or by the task prior to their use. This allows different devices and files to be used without program recompilation.

Devices

Input/Output support of peripheral devices such as card readers, line printers, etc. is provided by the various Device Drivers. This support is provided to enable a program to be written so that any device supporting the necessary operations may be used. Most drivers also provide support for device dependent operation so that special features and capabilities of the device may be used. In general Direct Access Files and peripheral devices may be accessed compatibly. Devices are referenced by a device mnemonic which is initially specified at system generation time. This name can be changed dynamically by the console operator or by an Executive Task (E-task).

Direct Access Files

All direct-access devices supported by OS/16 MT2 may be accessed through the OS/16 File Manager, which provides a substantial and powerful set of volume and file management services.

Data on a direct-access device is maintained as files on a named logical volume in a compatible manner with OS/32. Each volume contains all the information necessary to process the data on that volume. While a direct-access device is marked off-line, it is referred to by the device mnemonic associated with the device at system generation time. When a direct-access device is marked on-line, the name of the volume mounted on that device is associated with the device and used to refer to it Volumes must not be dismounted without marking the device off-line.

Before using a direct-access volume, it must be formatted by the Common Disc Formatter, program number 06-173, or the Common MSM Disc Formatter, program number 06-201, and initialized for OS/16 use by the OS/16 MT2 INITIALIZE operator command.

Volume Organization

Allocation of space on an OS/16 volume is made in a flexible way, in order to reduce the adverse effects of any defective sectors. Only one sector is specifically required to be valid, this is Sector 0, Cylinder 0. On this sector the system maintains the Volume Descriptor. All other structures on the disc are allocated by the system wherever feasible.

The Volume Descriptor consists of three fields as illustrated by Figure 1-3

Volume Name
Pointer to File Directory
Pointer to Allocation Map

The remainder of the Volume Descriptor is reserved for future expansion
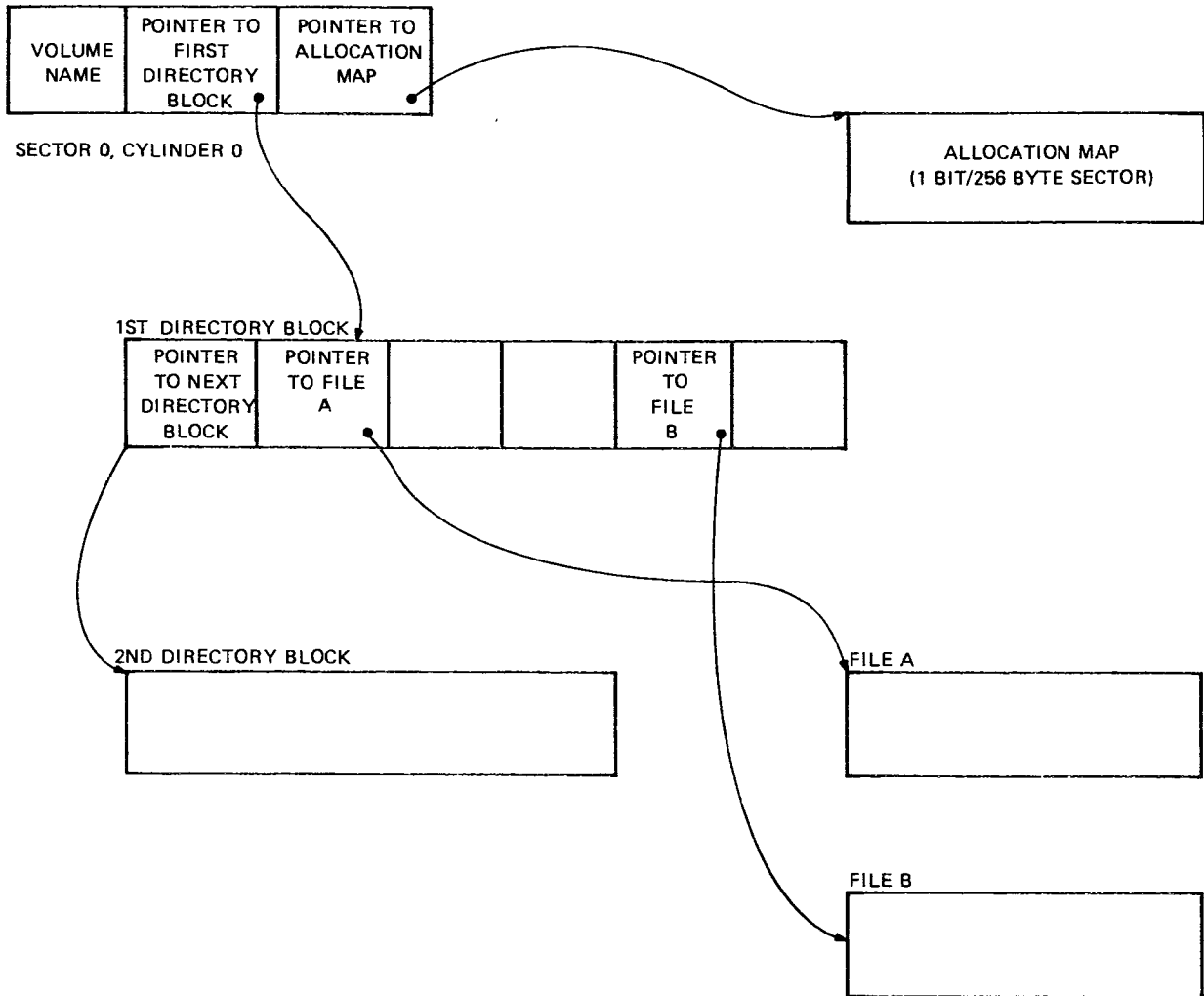
Figure 1-3. Volume Organization

Volume Name field contains a one to four-character ASCII volume identifier. This is the name by which the volume is known to the system.

Pointer to File Directory and Pointer to Allocation Map fields point to the first sectors of the File Directory and Allocation Map, respectively.

All data is initially placed in the Volume Descriptor by the OS/16 INITIALIZE command.

The Allocation Map is a bit-map having one bit for each sector on the volume. Since a sector occupies 256 bytes, the bit-map overhead is only 0.05% of the space on the volume. This map is used only to record allocated, unallocated and defective sectors. If a sector is allocated or defective, its corresponding bit in the Allocation Map is set to ONE; if unallocated, to ZERO.

The File Directory contains information needed by the system to process files recorded on the volume. An entry in the directory is made for each file.

The directory itself is organized as a chain of one-sector blocks. A directory block contains up to five file entries.

When a direct-access volume is initialized, a number of File Directory blocks are pre-allocated. When the first file on the volume is allocated, the first file entry in the directory is then marked as in use and the remaining four as not in use. As new directory blocks are needed, they are chained to the end of the File Directory.

Identification of Files

An OS/16 file is identified by a File Descriptor, which has three parts· volume name, file name, and extension.

The volume name is composed of from one to four alphanumeric characters, of which the first character must be alphabetic. This is the name of the volume on which one file resides.

The filename consists of from one to eight alphanumeric characters, of which the first must be alphabetic. This is the main identifier for the file, and may be anything the user chooses

The extension consists of up to three alphanumeric characters. It may consist of no characters at all, in which case it is considered to consist of blanks. The extension usually denotes the type of material on the file. It may be anything the user chooses; however, some specific extensions are used by OS/16 and by some OS utilities, and are assumed to have specific meanings. These extensions are·

OBJ	Absolute or Relocatable loader format
FTN	FORTRAN source format.
CAL	CAL assembly language source format.
BAS	BASIC source format.
CSS	Command Substitution System source format.
TSK	Task Image format.
ROL	System roll file
SPL	Printer spool file
PRT	Printer spool file
MAC	CAL Macro source format

The user may use any of these standard extensions, or may define others.

File Descriptors are written as follows.

VOLN:FILENAME.EXT

where VOLN is the volume name, FILENAME is the file name, and EXT is the extension. VOLN and EXT may be omitted when default names are assumed, such as the system volume and a blank extension.

A File Descriptor is also used to describe a device, in which case the VOLN field describes a device mnemonic rather than a volume name. The colon following the device mnemonic must be retained to avoid confusion with a file specification having a default volume name and extension The FILENAME and EXT fields are ignored for devices and should not be used.

File Organization

A file is a collection of related records From a programmer's point of view, a file is made up of logical records which may be of arbitrary length and structure, and are process-dependent. From the system's point of view, a file is made up of physical blocks which are of fixed length appropriate to the particular device and are process-independent. When a user program is written, the logical file structure must be considered because certain information is required at execution time by the I/O processor routines and therefore must be supplied by the user program. When a file is allocated, the manner in which the data is to be stored physically on the device must be specified.

OS/16 MT2 supports two file structures. Although these structures differ, in many cases the same data manipulations can be performed on one structure as another. The choice of file structure, in most applications, does not depend on the form of the data to be put in the file, but on the way in which the data is accessed. OS/16 MT2 file structures are each optimized for one specific form of access.

Indexed Files

The Indexed file is an open ended file structure consisting of two types of physical blocks: a chain of index blocks and individual data blocks pointed to by the index blocks. Figure 1-4 illustrates the relationship of the two types of blocks.

Both the index blocks and data blocks may be scattered on the volume. An Indexed file may be accessed sequentially or randomly. The index block structure reduces the number of physical accesses necessary to locate a specific data block.

Logical records may be added sequentially to the end of the Indexed file at any time. Existing logical records may be overwritten either sequentially or randomly, but cannot be deleted. For random access, logical records are referenced by a logical record number which is a number ranging from 0 to the number of logical records in the file minus 1. This logical record number indicates the order in which the logical record was added to the file.

When an Indexed file is allocated, the logical record size of the data to be contained is specified, as well as the physical block size of both the index blocks and the data blocks.

Access to the logical records on an Indexed file is via the Logical Buffered Access Method. which means that logical record length is independent of the physical block size specified for the data blocks. All blocking and deblocking is performed automatically by the OS via a buffer allocated in system space along with the File Control Block (FCB) when the file is assigned

Contiguous Files

The Contiguous file is a fixed-length file structure. All blocks of a Contiguous file are allocated contiguously on the volume as illustrated by Figure 1-4. The file size (in 256-byte sectors) is specified at the time of allocation, and all required space is reserved at that time Each sector (block) is considered a record by the system. Random reads and writes may access any record on the file. regardless of which records have been previously accessed This makes it possible to create a Contiguous file in a random fashion
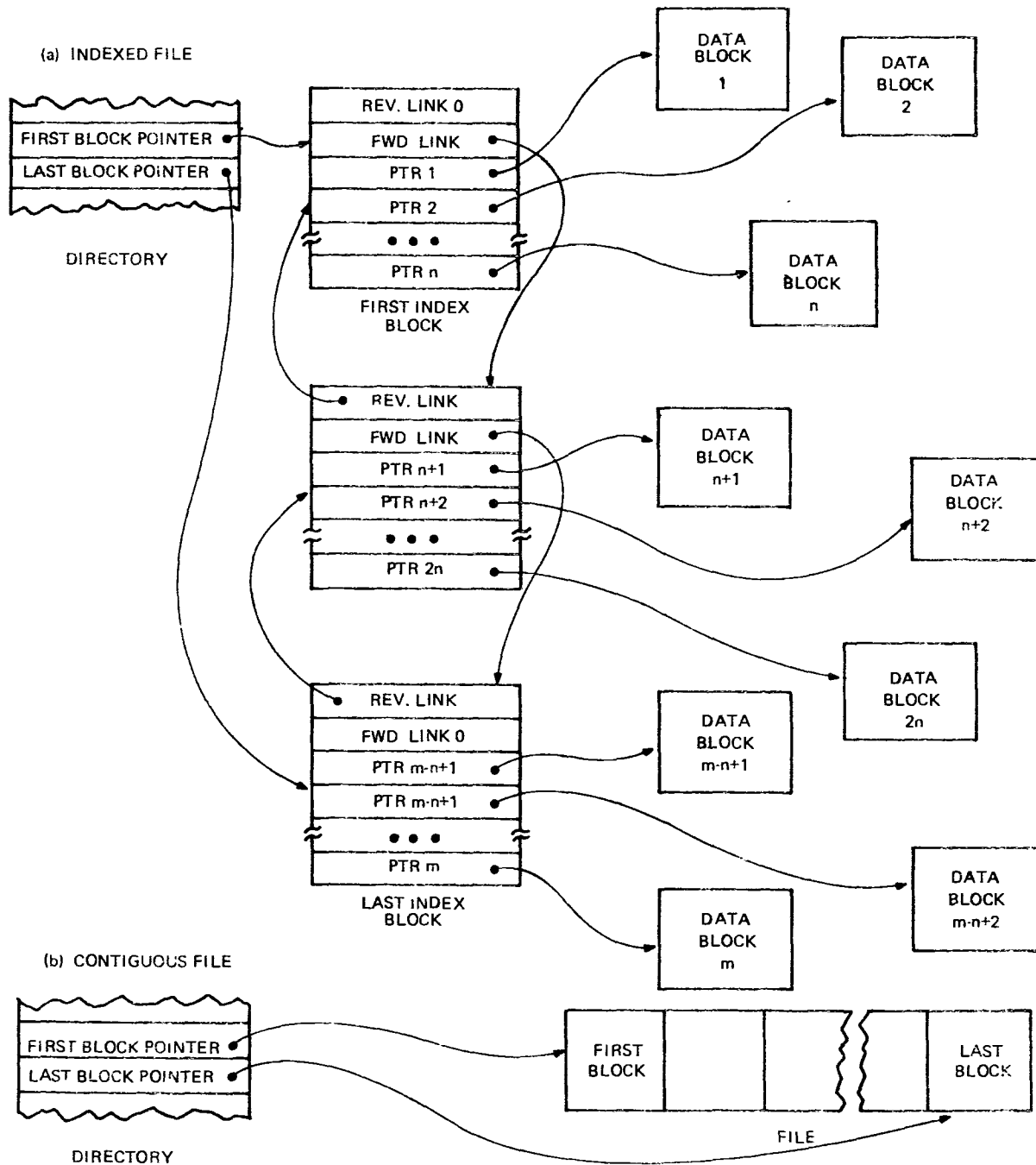
Figure 1-4. OS/16 MT2 File Structures

Contiguous file I/O is non-buffered and transfers a variable amount of data directly to or from the task's buffer. Although the user may transfer data in logical records of size greater or smaller than a sector, the appropriate sector number must be specified to position the file for random access. All transfers begin on a sector boundary and end whenever the specified number of bytes has been transferred.

The Contiguous file supports a pseudo filemark capability that gives it some of the characteristics of a magnetic tape device. A filemark is indicated by a halfword of X'1313' at the beginning of a record (block). Care should be taken to ensure that this datum is not inadvertently written at the beginning of a record if filemark operations are going to be used. The forward-file and backward-file operations, on a Contiguous file, function as they would on a magnetic tape. That is, the file is positioned forward or backward respectively until a filemark (X'1313') is found. The current record pointer is then left at the record following or containing this filemark, respectively The write-filemark operation results in writing X'1313' at the beginning of the current record

File Access Methods

OS/16 MT2 supports two methods of access to files: random and sequential. These methods may be intermixed without having to close and reassign the file. The chief mechanism used to implement these methods is the current record pointer.

The current record pointer is a number, ranging from zero to the number of logical records currently in the file. indicating the record to be read or written on the next sequential access to the file Each record is numbered in sequence, starting with zero

The current record pointer is adjusted in one of several ways·

1. It is set to zero by the following operations

   Rewind

   Backspace to filemark (except on Contiguous files where the record pointer is positioned at the record containing the filemark)

   Assigning (except for Write-only access)

2. It is set to the number of records in the file (the proper position to append new records) by the following operation:
   Assigning for Write-only access

   Forward to filemark (except on Contiguous files where the record pointer is positioned after the record containing the filemark).

3. It is decremented by one by a backspace record operation, unless the file is already positioned at its beginning.

4 It is incremented by one as follows:

   Forward record (unless already at end of file)

   Write filemark to a Contiguous file

   Sequential Read or Write to an Indexed file

5. A random Read or Write sets the current record pointer to a value one greater than the record read or written.

6. It is incremented by the number of sectors that must be accessed to satisfy a sequential Read or Write request to a Contiguous file.

Random Access

For random access, the user supplies the record number that is to be accessed. This record is found, the data transfer is performed, and the current record pointer is set to point to the next sequential record. If the user continues to use random access, the current record pointer may be ignored, since it is readjusted on every call. However, the user may wish to read or write a sequence of records, starting with a known record number. In this case, a single random call followed by a number of sequential calls may be used.

With an Indexed file, the user is somewhat restricted in the use of the random Write call. This call may be used to update any record currently in the file, or to append one record to the end of the file. If the record number specified is more than one record past the end of the file, the call is rejected. This means that a file must be expanded in a sequential manner. If the file has only five records, a sixth may be added but record number 100, for example, may not.

On Contiguous files there is no restriction on the use of the random Write or Read call. Any record within the file's allocation may be read or written.

Sequential Access

Sequential access is the simplest and most common access method The user performs a series of sequential Read or Write calls These cause records of the file to be read or written in sequence. The current record pointer is adjusted automatically at each access. The Rewind, Forward Record, Backward Record, Forward File and Backward File commands may be used for repositioning as described previously.

File and Device Protection

Files and devices may be protected in one of two ways: statically or dynamically.

Static Protection

Each file or device has associated with it two protection keys, one for Read access and one for Write access. Each key is one byte long and may have any value from X'00' to X'FF'.

If the values of the keys are within the range of X'01' to X'FE', the file or device may not be assigned for Read or Write access unless the operator or requesting task supplies the matching keys.

If a key has a value of X'00', the file or device is unprotected for that access mode Any key supplied is accepted as valid.

If a key has a value of X'FF', the file is unconditionally protected for that access mode. It may not be assigned for that access mode to any user task, regardless of the key supplied. An unconditionally protected file may be assigned to an Executive task, including the Command Processor.

Some examples of static protection follow.

Write Key	Read Key	Meaning
00	00	Completely unprotected.
FF	FF	Unconditionally protected.
07	00	Unprotected for Read, conditionally protected for Write (user must supply Write key = X'07').
FF	A7	Unconditionally protected for Write, conditionally protected for Read
27	32	Conditionally protected for both Read and Write.

The protection keys of a file are defined when the file is allocated, and may be changed by the console operator or by any task having that file assigned for exclusive Read/Write access (See Dynamic Protection)

The protection keys of a device are set to zero at system generation time, and may be changed by the console operator or by an E-Task.

Dynamic Protection

When a task has assigned a file, it may wish to prevent other tasks from accessing that file while it is being used For this reason, the user may ask for exclusive access privileges, either for Read or Write, at assignment time. This form of protection is called dynamic because it is only in effect while the file remains assigned

The access privileges are generally known by their abbreviations. These are:

SRO	Shared Read-Only
ERO	Exclusive Read-Only
SWO	Shared Write-Only
EWO	Exclusive Write-Only
SRW	Shared Read-Write
SREW	Shared Read, Exclusive Write
ERSW	Exclusive Read, Shared Write
ERW	Exclusive Read-Write

A file cannot be assigned with a requested access privilege if that assignment is incompatible with some other existing assignment of that file. For example, a request to assign a file for Exclusive Write-Only is compatible with an existing assignment of that file for SRO or ERO, but is incompatible with any existing assignment for other access privileges Table 1-1 shows compatibilities and incompatibilities between access privileges.

Write Protection

All files on a disc volume may be protected from Write operations by marking the disc on-line as a protected device. When a volume is marked as being Write protected, all assigns for access privileges other than Shared Read Only (SRO) and Shared Read/Write (SRW) are rejected with a privilege error: Shared Read/Write (SRW) is changed to Shared Read Only (SRO). If the hardware write protected feature of a disc is enabled, the volume must be marked on-line as a protected volume.

Null Device

If a task performs I/O requests via SVC 1 calls, but no actual transfer is desired (possibly during testing), the Logical Unit (LU) used in the transfer may be assigned to the Null device before starting the task. The OS/16 MT2 Configuration Utility Program configures every OS/16 MT2 with a Null device which may be referenced by the File Descriptor:

NULL:

Read requests to the Null device return EOF status with the specified buffer unchanged. All write requests return with normal status and a length of data transfers of zero.

TABLE 1-1. ACCESS PRIVILEGE COMPATIBILITY

LU 1 \ LU 2	SREW	EWO	SWO	SRW	SRO	ERO	ERSW	ERW
SREW	–	–	–	–	*	–	–	–
EWO	–	–	–	–	*	*	–	–
SWO	–	–	*	*	*	*	*	–
SRW	–	–	*	*	*	–	–	–
SRO	*	*	*	*	*	–	–	–
ERO	–	*	*	–	–	–	–	–
ERSW	–	–	*	–	–	–	–	–
ERW	–	–	–	–	–	–	–	–

* = compatible
– = incompatible

CHAPTER 2
TASK MANAGEMENT

TASKS

Tasks are controlled by OS/16 MT2 through two system control blocks: the Task Control Block (TCB), and the User Dedicated Locations (UDL). The TCB is maintained in system space and therefore cannot be modified by a user task. The User Dedicated Locations (UDLs) are maintained in the task's own allocation, therefore providing a measure of self-control to the task. This chapter describes the data structures and services related to task management.

Most of the services described here are invoked through the Intertask Communication Supervisor Call (SVC 6) described in Chapter 3.

Interrupts and Traps

In OS/16 MT2, as in most operating systems, all interrupts at the processor level, both external and internal, are handled by the operating system. OS/16 MT2 provides an interrupt facility at the task level known as the task-handled trap facility. This facility permits a task to be interrupted out of its normal execution sequence for any one of a variety of causes, both hardware and software-generated. This facility also allows one foreground task to pass a parameter to another foreground task via the receiving task's Task Queue, thus permitting the receiving task to determine the software defined reason for the interrupt. A task handled trap may occur for the following reasons:

> Power Restoration
> Addition of an item to the Task Queue
> Completion of an I/O proceed request
> External Interrupt from a Trap Generating Device
> Completion of a specified time delay
> Message received from a task

Task Status Word (TSW)

Traps, and additions to a task's Task Queue are controlled through the task's Task Status Word (TSW), which is the task level analog of the 16-Bit Processor Program Status Word (PSW). This status word is used to enable or disable the various traps, enable or disable additions to the Task Queue, and to save the location counter and condition code of a task at the time of a trap. TSW bit assignments are shown in Figure 2-1. The current Task Status Word (TSW) is maintained by the system for each task in its Task Control Block (TCB). The initial value of a task's TSW is set by the user (or by default) at task establishment time.
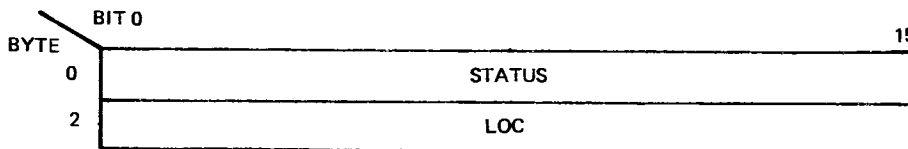
Figure 2-1. Task Status Word (TSW)

Task Status Word is 32 Bits. First halfword (bits 0-15) contains status defined in Figure 2-2. Second halfword (bits 16-31) contains the current location counter as in a PSW.

BIT	HEX MASK	NAME	DESCRIPTION
0	8000	W	Tr..p Wait Task suspended until a task trap occurs
1	4000	P	Po ver Restoration Trap Enable. A task trap occurs on completion of system power restore sequence
2-3			R(served Must be Zero
4	0800	Q	T.·k Queue Service Trap Enable A task trap oci urs when an item is added to the Task Queue Also a trap occurs if a TSW is loaded with this bit set and the Task Queue is not empty
5			Reserved Must be Zero
6	0200	D	Enable Task Queue entry on interrupt from a connected Trap Generating Device.
7	0100	T	Enable Task Queue entry on Queue Parameter (SVC 6) request directed at this task.
8	0080	Z	Enable Task Queue entry on completion of a time interval
9	0040	O	Ei able Task Queue entry on completion of I/O proceed request.
10			Reserved. Must be Zero.
11	0010	E	Enable Task Queue entry on send message
12-15	000F	CC	Condition Code (as in PSW)

Figure 2-2. Task Status Halfword

User Dedicated Locations (UDL)

A minimum of 36 bytes at the start of a task's program address space must be reserved for the User Dedicated Locations A 68 byte UDL is required for a task which uses Single Precision Floating Point, and 132 bytes for a task which uses Double Precision Floating Point (with or without single precision). The UDL contains system pointers, TSW swap areas, and other data used for communication between the operating system and the task, plus floating point register save areas as required. A map of the UDL is given in Figure 2-3. A task's UDL will, by default, be reserved and initialized to zero by TET/16 when the task is established The UDL can then be set up dynamically by the task at run-time, if necessary Alternatively, the UDL area can be assembled as part of the task, or as a sub-program to be included as the first part of the task The program containing the UDL must be biased at the first address of the specified partition during task establishment See Chapter 5 of the *OS/16 MT2 Operator's Manual* for further information.

CTOP — This location contains the address of the highest halfword in the task's allocated memory (program segment). This address does not reflect the possible use of a resident library or Task Common partition.

COMBOT — This location is provided for compatibility with previous INTERDATA operating systems. COMBOT is always set equal to CTOP in OS/16 MT2.

UTOP — This location contains the address of the first halfword following the user program This value may be modified by the use of Get and Release Storage Supervisor Calls (SVC 2). UTOP always contains an address aligned on a halfword boundary.

UBOT	This location contains the lowest address in the user program segment.
TSW Swap areas	These are provided for Power Restoration and Task Queue Service Traps. They are each four halfwords in length. The first two halfwords of each location serve to save the previous TSW when a trap occurs; the new TSW is taken from the second two halfwords.
A(TASKQ)	This location contains the address of the Task Queue. It must be set up by the program prior to enabling any entries to the Task Queue. If the contents of this location is zero, no Task Queue entries are made, regardless of the state of the TSW enable bits for Queue entry.
A(MESSAGE BUFFER)	This location contains the address of a 74 byte buffer beginning on a halfword boundary. This field must be set up by the task prior to receiving a message If the content of this location is zero, no message can be received regardless of the TSW queue entry enable bit.

0 (0)	2 (2)
CTOP	COMBOT
4 (4)	**6 (6)**
UTOP	UBOT
8 (8) A(TASKQ)	
10(A) POWER RESTORE OLD TSW SAVE AREA	
14(E) POWER RESTORE NEW TSW	
18(12) TASK QUEUE SERVICE OLD TSW SAVE AREA	
22(16) TASK QUEUE SERVICE NEW TSW	
26(1A) A(MESSAGE BUFFER)	28(1C) RESERVED
30(1E) RESERVED	
36 (24) SINGLE PRECISION FLOATING POINT REGISTER SAVE AREA	
68(44) DOUBLE PRECISION FLOATING POINT REGISTER SAVE AREA	

132(84)

Figure 2-3. User Dedicated Locations (UDL)

TASK-HANDLED TRAPS

When a condition occurs that causes a trap, a TSW describing the routine to handle the specific type of trap is loaded from the appropriate new TSW in the task's User Dedicated Locations (UDL), the task's current TSW is saved in the UDL and control is passed to the routine to handle the trap. It is the trap routine's responsibility to save all general or floating point registers used within the routine prior to servicing the trap. A Load TSW Supervisor Call (SVC 9) is used to load the saved current TSW, thus returning control to the normal execution sequence.

If a task is in any Wait state other than Trap Wait, a trap does not actually occur until the task has left that Wait state. The TSW swap occurs upon detection of the condition causing the trap; however, the task remains in the Wait state until the reason for that wait is removed. Multiple TSW swaps may occur before the task leaves the Wait state, if so enabled. It is the responsibility of the user to assemble, or dynamically prepare, the desired new TSWs in the UDL for each type of trap, if the task trap facility is to be used.

Task Queue Service Traps

Since several trap-causing conditions may occur before the first trap is handled by the task, the Task Queue facility is provided to allow for queuing of trap information during periods when the task is unable to service a trap.

The following trap-causing conditions cause an item to be added to the task's Task Queue:

> Addition of a parameter to the Task Queue
> Completion of an I/O proceed request
> External Interrupt from a Trap Generating Device
> Termination of a specified time delay
> Message received from a task

As shown in Figure 2-2, each of these reasons can be independently disabled by setting the appropriate enable bit to ZERO

The Task Queue is a "circular list" in the standard INTERDATA list-processing format. This list may be of any size desired, and may be located anywhere within the user program. It should be large enough to contain all parameters which may be queued at any one time. If the queue is full, attempts to add to the queue are rejected by SVC 6. A pointer to this list must be placed in the User Dedicated Locations by the program before enabling Task Queue entries.

The system always adds items to the bottom of the Task Queue

Table 2-1 lists the parameters that are added to a Task Queue by the OS If the Task Queue is full, any entries the system attempts to queue are lost.

TABLE 2-1. TASK QUEUE ENTRY TYPES

TYPE	CONTENTS
Device Interrupt	Parameter associated with device in SVC 6 connect time
SVC 6 Queue Parameter	Parameter specified in Add Parameter To Task Queue call (SVC 6)
I/O Proceed Complete	Address of I/O Call (SVC 1) parameter block.
Interval Completion	Parameter specified at interval initiation (SVC 2 Code 23)
Task Message	Address of message buffer

Power Restoration Traps

A task may wish to be informed if the processor has executed a power fail/restore sequence. Upon power failure, OS/16 MT2 saves all necessary data so that system operation can resume when power is restored Although the system is able to resume normally, the same is not always true of tasks, since I/O requests may have been aborted or a time interval may have been missed.

Any I/O operation in progress when power fails is aborted. If the operation is on a direct-access device, the system is able to retry it when power is restored; however, on other than direct-access devices, a retry is generally impossible. These I/O operations, therefore, return an unrecoverable error status to the task.

If a task has real-time considerations, power restoration may indicate that an appreciable amount of time elapsed while power was down. This fact, and the lack of knowledge at the task level as to how long the power outage lasted, may pose substantial difficulties to a real-time task.

The normal course of events, in case of power outage, is to pause all tasks in the system. However, a task may choose to take a Power Restoration trap, instead of being paused.

If this is the case, the task should keep the P bit (Bit 1) set in its current TSW. Then, whenever there is a power outage, a TSW swap occurs upon restoration of power, and the task can go about the business of recovering from the power outage.

Trap-Generating Devices (TGD)

In certain applications, it is desirable for a task to respond to interrupts from some external device. OS/16 MT2 provides a set of facilities to allow this to be done.

Certain drivers, in particular the Eight-Line Interrupt Module driver, are capable of adding a parameter to a task's Task Queue in response to an interrupt from the device. The addition to the Task Queue can cause the task to take a trap, if enabled. For this reason, these devices are called Trap-Generating Devices (TGD) and their drivers are called TGD drivers.

Most devices, such as TTY, Card Reader, et cetera, are not capable of being used in this way. The only driver offered with OS/16 MT2 that supports TGD functions, is the Eight-Line Interrupt Module driver.

The functions provided by OS/16 MT2 for the handling of TGDs implement the entire ISA (Instrumental Society of America) proposed standards for process control. These functions are:

Connect	attaches a TGD to a task
Thaw	enables interrupts on a TGD
SINT	simulates an interrupt on a TGD (Addition to ISA standard)
Freeze	disables interrupts on a TGD
Unconnect	detaches a TGD from a task

INTERRUPTS

Interrupts handled by the OS/16 MT2 Executive are: Floating Point and Fixed Point Divide Arithmetic Faults, Illegal Instruction, Machine Malfunction, and Memory Protect Fault.

Floating Point and Fixed Point Divide Arithmetic Faults

When an Arithmetic Fault interrupt occurs, the task causing it may either be paused or continued, depending on selected task option. In either case, a message is output to the system log. The task has the further option of disabling arithmetic fault interrupts entirely, by the use of the Set Status Supervisor Call (SVC 2 Code 4). If an Arithmetic Fault occurs in system code execution, the crash handler is entered.

Illegal Instruction

If the illegal instruction was executed within system code, the crash handler is entered; if the illegal instruction was executed within user code, a message is output to the system log and the task is paused.

The previous statement does not apply if the illegal instruction interrupt was caused by the execution of a floating-point, multiply/divide or list instruction on a system using software traps. In that case, the appropriate trap routine is entered and the execution of the instruction is emulated.

Machine Malfunction

The machine malfunction interrupt occurs on memory parity error, power failure, and power restoration

A memory parity error causes a system crash if it occurs within system code; otherwise, a message is output to the system log and the active task is paused Memory parity errors may occur when addressing non-existent memory in a processor with the parity feature installed.

Power failure causes the system to prepare for an orderly shutdown and to prime itself to await the machine malfunction interrupt that is to occur on power restoration

On power restoration, a message is logged requesting the console operator to reset all I/O devices (such as disc, which may not be ready or may come up in a Write-disabled state). When this message is acknowledged by the operator, all pending I/O requests are terminated, except those on direct-access devices, which are retried. All tasks in memory are then paused, except for those with a TSW enabling a task-handled trap on power restoration. The SET TIME command should be used to reset the clock as soon as possible following power restoration.

Memory Protect Fault

This interrupt occurs when a task attempts to violate the conditions of memory protection imposed by the Memory Protect Controller (MPC). A message is output to the system log and the errant task is paused.

TASK STATUS AND OPTIONS

Task Status and Options halfwords are maintained by the system in the Task Control Block (TCB) Task Options, in general, define overall characteristics of the task while the Task Status defines the state of the task. Tables 2-2 and 2-3 define the meanings of the Status and Options halfwords.

TABLE 2-2. TASK OPTIONS

BIT	HEX MASK	MEANING
0	8000	Executive Task (E-task) if 1 User Task (U-task) if 0
1	4000	Arithmetic Fault Continue if 1 Arithmetic Fault Pause if 0
2	2000	Task uses Single Precision Floating Point registers if 1 Task does not use Single Precision Floating Point registers if 0
3	1000	Task is Memory Resident if 1 Task is non-resident if 0
4	0800	Task uses BOSS/DOS/RTOS SVC 1 format if 1 Task uses OS/16 MT2 SVC 1 format if 0
5	0400	Task is the background task if 1 Task is a foreground task if 0
6	0200	SVC 6 ignored (Background only) if 1 SVC 6 illegal (Background only) if 0
7	0100	Task is rollable if 1 Task is not rollable if 0
8		Reserved for system use
9	0040	Task uses Double Precision Floating Point registers if 1 Task does not use Double Precision Floating Point registers if 0
10-15		Reserved for system use

TABLE 2-3. TASK STATUS BITS (TCB)

BIT	HEX MASK	MEANING IF SET
0	8000	Dormant
1	4000	I/O Wait
2		Reserved
3	1000	Roll Wait
4		Reserved
5	0400	Console Wait (Paused)
6	0200	Time Wait
7	0100	Trap Wait
8	0080	SVC 2 Wait
9	0040	SVC 5/6 Wait
10	0020	SVC 7 Wait
11	0010	Reserved
12	0008	DCB Wait (File I/O Wait)
13-15		Reserved

CHAPTER 3
SUPERVISOR CALLS

SVC INSTRUCTIONS

The SVC instruction enables a user task to communicate with the operating system and to use the software facilities provided. Execution of an SVC instruction at the assembly language level causes an internal interrupt which is processed by the Executive of OS/16 MT2. Higher level languages provide statements which generate SVC instructions.

The general form of an SVC instruction is:

   SVC n,P

Where·   n   specifies the particular SVC
         P   represents a parameter or the address of a parameter block which further defines the request. Parameter blocks must be aligned on a halfword boundary.

Valid SVC Calls

OS/16 MT2 provides all SVC instructions supported by OS/32 MT except for the OS/32 AIDS Support Supervisor Call (SVC 14), although not every option of each OS/32 MT SVC is supported. Table 3-1 summarizes the SVCs supported by OS/16 MT2.

TABLE 3-1. OS/16 MT2 SUPERVISOR CALLS

SVC	CODE*	FUNCTION
1		I/O Request
2	1	Pause
	2	Get Storage
	3	Release Storage
	4	Set Status
	5	Fetch Pointer
	6	Unpack
	7	Log Message
	8	Interrogate Clock
	9	Fetch Date
	15	Pack Numeric Data
	16	Pack File Descriptor
	17	Mnemonic Table Scan
	18	Move ASCII Characters
	19	Peek
	23	Timer Management
3		End of Task
5		Fetch Overlay
6		Intertask Coordination
7		File Management Services
9		Load TSW

*The code number appears in the SVC 2 parameter block

SVC Errors

OS/16 MT2 processes errors in an SVC instruction by issuing a message to the system console and pausing the task

The two possible error condition, and their causes are-

1. Illegal SVC which indicates an illegal SVC number (n), or invalid parameter specified in the parameter block.

2. Invalid Address in SVC which indicates an invalid parameter block address (not in task's allocation, or not on a halfword boundary), or an invalid address specified in a parameter block.

In either case the address of the errant SVC instruction is displayed in the message

SVC 1 – INPUT/OUTPUT REQUEST

SVC 1 is used by a task to perform all I/O requests. The parameter block format required to define the request is:

0(0) FUNCTION CODE	1(1) LOGICAL UNIT
2(2) DEVICE INDEPENDENT STATUS	3(3) DEVICE DEPENDENT STATUS
4(4) BUFFER START ADDRESS	
6(6) BUFFER END ADDRESS	
8(8) RANDOM ADDRESS (4 BYTES)	
12(C) LENGTH OF DATA TRANSFER	
14(E) COMMUNICATIONS EXTENDED OPTIONS (ITAM/16 ONLY)	

18 (12)

This parameter block must be on a halfword boundary and must be in a writable segment of the program address space. An SVC 1 call may be coded as follows:

SVC	1,PARBLK	
.		
.		
.		
ALIGN	2	
DB	X'function code'	FUNCTION CODE
DB	X'logical unit'	LOGICAL UNIT
DS	2	STATUS
DC	A(START)	START ADDRESS
DC	A(END)	END ADDRESS
DC	Y'RANDOM'	RANDOM ADDRESS
DS	2	LENGTH OF DATA TRANSFER
DC	Y'COMOPT'	COMMUNICATIONS OPTIONS

Data Transfer Requests

The function code field for a data transfer request is defined in Table 3-2.

TABLE 3-2. SVC 1 DATA TRANSFER FUNCTION CODE

BIT	ALIGNMENT	MEANING
0	X. . ..	This bit must be ZERO to indicate a data transfer request.
1-2	.XX. ...	Read-Write bits The meaning of these two bits is modified by bits 3-7 to control the transfer Basically the values are: 10 - Read request 01 - Write request 11 - Test and Set request (see Chapter 8 on Contiguous Files) 00 - Wait only or Test I/O Complete
3	. .X ..	ASCII/BINARY bit. This bit indicates the type of formatting requested if Bit 7 is reset. 0 - indicates ASCII formatting 1 - indicates binary formatting If Bit 7 is set, this bit must be ZERO
4	. . X...	PROCEED/WAIT bit. This bit indicates the action to be taken after the I/O has been initiated. 0 - Proceed Indicates that control is to be returned to the task after initiation of I/O. 1 - Wait. Indicates that the task is to be put into I/O Wait until the data transfer is complete.
5	. ... .X. .	SEQUENTIAL/RANDOM bit. 0 - Sequential. Indicates the next logical record is to be accessed. 1 - Random. Indicates the logical record specified by the RANDOM field is to be accessed.
6	.... ..X.	UNCONDITIONAL PROCEED bit. 0 - indicates the task is to be put into I/O wait until the requested device/file is free. At that time the request is processed. 1 - indicates that the request is to be rejected with a condition code of $X'F'$ if the requested device/file is not free
7	X	FORMATTED/IMAGE bit. 0 - indicates that the request is to be formatted according to the device/file and the setting of Bit 3. For communications devices indicates default extended options to be used 1 - indicates that no formatting is to be performed (Image mode) For communications devices indicates Extended Option field applies.

In general, the request is defined by the logical OR of the function code bits. The ZERO setting of each bit is valid for all devices/files. If any invalid ONE setting of a bit is specified, the request is rejected by the OS as an illegal function (see STATUS byte definition).

For example, a function code of $X'5C'$ specifies a request for:

Read ($X'40'$)
Binary Formatting ($X'10'$)
Wait I/O ($X'08'$)
Random Access ($X'04'$)

The full SVC 1 parameter block, except the communications extended options field, must be reserved for data transfer requests. Wait only and Test I/O complete requests make use of the function code, logical unit, and status fields only, so the remaining fields may be redefined and used by the task (e.g., for storing constants, etc.).

Command Function Requests

The function code for a command function request is defined in Table 3-3

TABLE 3-3. SVC 1 COMMAND FUNCTION CODE

BIT	ALIGNMENT	MEANING
0	x..	This bit must be set to indicate a command function request. This bit alone requests Halt I/O
1	x..	Rewind
2	. x ..	Backspace Record
3	. x .	Forward Space Record
4	..x. .	Write File Mark (EOT for communications device.)
5	. ...x .	Forward Space File
6	....x.	Backspace File
7	x	Reserved for driver dependent functions

The implementation of all commands is device-dependent and is explained for each driver in Chapter 8. The implementation of command functions for direct-access files is also explained in Chapter 8.

When multiple bits in the command function code byte are set, the following principle applies

> The function code is scanned from left to right The leftmost bit found that is meaningful to the device assigned to the specified logical unit indicates the function to be performed.

If no valid function is indicated by the function code, the system returns immediately to the user, with normal status Note that this condition is not considered an error The user may find out which command functions are supported on any Logical Unit by means of the Fetch Attributes (SVC 7) call.

Most drivers do not make use of the fields following the Device Dependent Status field (for commands) If the user is certain that no special drivers are to be used by the task, these fields may be put to other uses, e.g., the storing of constants, etc However. for a fully device-independent program, it would not be appropriate to use these fields.

Logical Unit (LU)

In order to provide device independent I/O, all I/O requests are directed to a Logical Unit (LU). An LU is a number from 0 to a SYSGENed maximum (up to 31) which describes an entry in the task's LU table. The particular device or file desired must be assigned to the specified LU via operator command (see OS/16 MT2 Operator's Manual). or SVC 7 call prior to executing the SVC 1 call. If an invalid or unassigned LU is specified, the call is rejected. If no operation is desired, the specified LU should be assigned to the Null device.

Error Status (Device Dependent and Device Independent Status)

The system returns the status of the requested function in the Device Independent Status byte. This Status byte is set to indicate the general type of error that occurred If there was no error, it is set to zero. In addition, the processor's Condition Code is set to zero. In the case of a rejected Unconditional Proceed call, the Condition Code is set to X'F'

The Device Dependent byte is set to zero along with the Device Independent Status byte, if there is no error. In case of error, a code further explaining the error status is returned in the Device Dependent Status byte.

The system does not modify the contents of the STATUS bytes until the requested function is complete or an error has been detected. While the function is in progress, the previous contents of these bytes are left unchanged

Specific interpretations of the error codes as they apply to devices and files are explained in Chapter 8. The general definition of the status bits is given in Table 3-4.

TABLE 3-4. SVC 1 DEVICE INDEPENDENT STATUS BYTE

BIT	MEANING IF SET TO ONE	BINARY	HEXADECIMAL
0	Always 1 for error status		
1	Illegal Function	1100 0000	X'C0'
2	Device Unavailable	1010 0000	X'A0'
3	End of Medium	1001 0000	X'90'
4	End of File	1000 1000	X'88'
5	Unrecoverable Error	1000 0100	X'84'
6	Recoverable Error	1000 0010	X'82'
7	Illegal or Unassigned LU	1000 0001	X'81'

The SVC 1 Device Dependent Status byte is defined as follows:

          X'82'   I/O terminated by time-out
          X'81'   I/O terminated by Halt I/O

If no dependent status is available, the Device Dependent Status byte contains the device number.

In general, for a data request, if Illegal Function, Device Unavailable, or Illegal LU status is indicated, then no data was transferred. Otherwise, some data may have been transferred. Device Unavailable may indicate that the device is physically inoperative. If the device becomes unavailable after a transfer is started, Unrecoverable Error is set to indicate the possibility that data was transferred.

The Illegal Function bit is set whenever the system cannot accept the SVC 1 Function Code byte as specified, for a data transfer. This may be for one of the following reasons:

1. A modifier was specified that is not supported by the device; e.g., Binary on a CRT.

2. A function was specified that is not supported by the device; e.g., Read on a Line Printer.

3. A function was specified that is not supported by the access privileges granted at Assign time; e.g., Read on a file that is assigned for Write-Only access.

Buffer Address

For data transfer requests, the buffer must be specified by the buffer start and buffer end address fields. The buffer start address is the first byte in the buffer and the buffer end is the last byte in the buffer (not the first byte after the buffer).

All buffers must be on a halfword boundary and must be fully contained in the same logical segment. Buffers used in Read requests must be in a writable segment.

An Invalid Address in SVC error occurs if

          Buffer is not on halfword boundary
          Buffer start and end are not in same logical segment
          Buffer end address is less then buffer start address
          Buffer for Read request is in Write-protected logical segment

Random Address

The Random Address field is 4 bytes long and is used to specify the logical record number (starting at 0) to be accessed on data transfer requests if function code Bit 5 is set The interpretation of this field is device or file dependent and is defined in Chapter 8

Length of Data Transfer

This field is used to return the actual number of bytes transferred during a data transfer request This field is most useful when dealing with variable length record devices, such as magnetic tape. This field is undefined if an error status is returned. (See note in the section entitled Test I/O Complete.)

Unconditional Proceed

Unconditional proceed I/O is used when a task does not wish to wait for the requested device or file to be free I/O requests are coordinated by the system so that only one I/O request may access any device at a time. If Unconditional Proceed is not requested, and the specified device is in use at the time of the data transfer request, the calling task is placed in I/O Wait until the device is free At that time, the I/O request is initiated. If Unconditional Proceed is specified and the specified device is in use, the SVC 1 request is rejected. Error status is set to zero (normal), and the condition code is set to X'F'. The calling task may then retry the request at a later time. If the specified device is not in used, the setting of the Unconditional Proceed Bit has no effect on the request.

Wait/Proceed I/O

Once an I/O request has been initiated (that is, the specified device is free), an I/O Proceed call (Bit 4 of function code reset) causes control to be returned to the task so that the task may execute concurrently with the transfer. The status is not set until completion of the I/O (except for Illegal function and Illegal LU which are detected before transfer initiation)

The status of the request may be checked by

> Monitoring the status field in the parameter block
> Taking a task handled trap on completion
> Issuing a Wait Only request to the same LU to wait for completion
>
> Using Test I/O complete on the same LU to monitor the completion

An I/O and Wait call (Bit 4 of the function code set) cause the calling task to be placed in I/O wait until completion of the request.

Wait Only

A Wait Only request (function code X'08') causes the task to be placed into I/O wait until the completion of a previous I/O Proceed request to the specified LU. If there is no uncompleted I/O by this task to the specified LU, control is returned immediately. This call makes use of the FC, LU, and STATUS fields of the SVC 1 parameter block. Illegal LU is the only error status returned by this call The status of the I/O being completed is returned in the parameter block associated with the original I/O Proceed call and not in the Wait Only call parameter block

Test I/O Complete

A Test I/O complete call (function code X'02') returns with a condition code of X'0' if there is no uncompleted I/O Proceed call to the specified LU by the task. If there is an uncompleted I/O Proceed, the call returns a condition code of X'F'. Illegal LU is the only error status returned by this call. The status of the I/O being tested is returned in the parameter block associated with the original I/O Proceed call.

<div align="center">

NOTE

For compatibility with older 16-bit Operating Systems, a task may use the compatibility option and use BOSS/DOS/RTOS SVC 1 parameter block format. Under this option, the length of data transfer field is unused, and the random address field is one halfword in length. For a wait-only call, the FC, LU and STATUS fields are used.

</div>

Halt I/O Command

Halt I/O allows a task to cancel an outstanding I/O and proceed request. This is particularly useful on an interactive terminal device, without the ability to Halt I/O, an outstanding read request must be satisfied before any other I/O can be started on that device.

The Halt I/O command is supported on all interactive terminal devices currently supported on the system; TTY Keyboard/printer, Carousel, CRT devices on a Current Loop or PASLA interface.

When a Halt I/O command is received, the system first verifies that it is a valid request:

    1   The logical unit (LU) must be assigned
    2.   The device must support Halt I/O
    3.   The LU must currently have a proceed I/O request outstanding.

If it is found to be invalid, an error status is returned to the Halt I/O request parameter block If the request is valid, the I/O operation is scheduled for termination and a status of zero is returned to the Halt I/O parameter block. The driver aborts the I/O and sets the status of the original I/O request to X'8281'.

The Halt I/O parameter block should not be the same one originally set up to request the I/O, as a result of the Halt I/O request, status will be returned to both parameter blocks. After the SVC 1 is issued, the Halt I/O status should first be checked to ensure that the system considers the request valid. The possible values are:

    X'0000'   The requested I/O termination has been scheduled
    X'8100'   The LU was not assigned
    X'82dn'   No I/O on-going for this LU
    X'C0dn'   Device does not support Halt I/O

where

    X'81' = unassigned LU
    X'82' = recoverable error
    X'C0' = illegal function
    dn   = device number

The task will take a trap when the I/O completes if Proceed I/O traps and Task Queue Entries are enabled; exactly as for any other Proceed I/O completion. The parameter added to the queue is the address of the original data transfer parameter block. if traps are not enabled, the task can wait for completion of the Halt I/O with an SVC 1 wait only request.

On completion due to Halt I/O request, the status returned is.

    X'8281'

where

    X'82' = recoverable error
    X'81' = device dependent status indicating Halt I/O request

Note that support for the Halt I/O function is a SYSGEN option. On any system configured without this support, no device will support Halt I/O, and an error status of X'C0' indicating illegal function will be returned on any Halt I/O request.

SVC 2 — GENERAL SERVICE FUNCTIONS

This SVC provides a number of general service functions. These functions relate to the task's communication with the console operator, to memory allocation, to text-processing and command-processing functions, and to timer management.

All SVC 2 calls require a parameter block. The general structure of an SVC 2 parameter block is as follows

0 (0) 1 (1)	2 (2)
OPTIONS CODE	
(other data as required)	
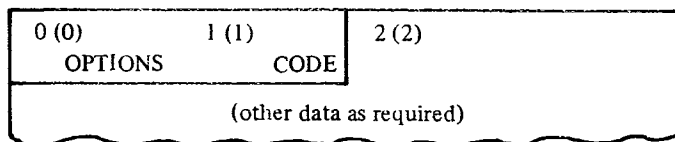
The CODE byte is a decimal number which may range from 0 to 255. Not all of the 256 possible codes are implemented at this time, unrecognized codes are rejected as illegal SVCs. The SVC 2 codes implemented in OS/16 MT2 are shown in Table 3-1.

The OPTIONS byte is generally used to further modify the conditions of the call Options may be used by individual SVC 2 codes as required, these are described in subsequent sections For those functions for which no options are defined, the options byte is ignored.

The remainder of the parameter block may be as long or as short as required for proper operation of the SVC 2 code. All SVC 2 parameter blocks must be aligned on a halfword boundary.

Code 1 — Pause

This call is used to place a task in a Console Wait state A PAUSE message is issued to the system log If the operator enters a CONTINUE command at the system console device, the program is restarted at the instruction immediately following the supervisor call. The parameter block format is as follows

	ALIGN	2
PARBLK	DB	0,1 NO OPTIONS

Incomplete I/O Proceed requests continue to completion normally even while the task is in the PAUSEd state

Code 2 — Get Storage

This call is used to provide temporary storage locations for certain subroutines called by a program, in particular FORTRAN Run-Time Library subroutines This call does not increase the size of the program's memory allocation, but obtains locations from the program's current allocation. The parameter block format is as follows:

	ALIGN	2	
PARBLK	DB	OPTION,2	DESIRED OPTION
	DC	H'REG'	ADDRESS OF REGISTER
	DC	H'SIZE'	NUMBER OF BYTES

Options allowed are

$\qquad$ X'00'   Get specified number of bytes
$\qquad$ X'80'   Get all allocated storage

If option X'00' is specified, the system adjusts the task parameter, UTOP, upwards by the number of bytes requested. The starting address of the storage obtained is returned in the designated register. Subsequent calls with this option obtain new areas. Since UTOP is maintained on a halfword boundary, requests are rounded up to the nearest halfword boundary.

If option X'80' is specified, the system sets UTOP equal to the task parameter CTOP+2, thus making available all of the task's current allocation. The starting address of the storage obtained is returned in the designated register and the number of bytes obtained is placed into the parameter block's SIZE parameter. The parameter block must be in a writable logical segment if this option is selected.

If the call is successful, the Condition Code is set to zero. If more storage is requested than is currently available, or a negative size is specified, an address of zero is returned, UTOP is not changed and the 'V' bit of the Condition Code is set (CC=4).

Code 3 — Release Storage

This call is the inverse of the Get Storage call (Code 2). It is used to release storage previously obtained. The format of the parameter block is as follows:

	ALIGN	2	
PARBLK	DB	0,3	NO OPTIONS
	DC	H'SIZE'	NUMBER OF BYTES TO BE RELEASED

This call does not reduce the program's memory allocation. The pointer UTOP is adjusted downwards by this call, but not below UBOT. Since UTOP always points to a halfword boundary, it is rounded up to the nearest halfword if the number of bytes released is not a multiple of 2. If the call would adjust UTOP below UBOT, or if a negative size is specified, UTOP is not changed and the Condition Code 'V' bit is set (CC=4), otherwise the Condition Code is set to zero.

Code 4 — Set Status

This call allows the user to modify the Floating Point Arithmetic Fault and Fixed Point Divide Interrupt Enable bits (AF) and the Condition Code (CC) of the PSW. Two options are provided. the first option specifies that all allowable bits be modified; the second option specifies that only the Condition Code be modified. The format of the parameter block is·

	ALIGN	2	
PARBLK	DB	OPTION,4	DESIRED OPTION
	DB	AF,CC	NEW STATUS, CONDITION CODE

The options allowed are:

X'00'	Modify Status and Condition Code
X'80'	Modify Condition Code only

The AF byte parameter indicates how arithmetic faults are to be handled and the valid combinations are:

X'00'	Disable Arithmetic Faults
X'10'	Enable Fixed Point Divide Faults
X'04'	Enable Floating Point Arithmetic Fault
X'14'	Enable both Fixed Point Divide and Floating Point Arithmetic Faults

The CC byte parameter has the form:

X'0x'	where x is to replace the Condition Code.

Code 5 — Fetch Pointer

This SVC is used by a task that wishes to determine the extents of its memory allocation or to modify its UDL It returns a pointer to the base of the task's User Dedicated Locations (UDL)

The format of the parameter block is as follows:

	ALIGN	2	
PARBLK	DB	0,5	NO OPTIONS
	DC	H'REG'	REGISTER SPECIFICATION

No options are recognized. The REG field contains the number of one of the user's general registers; the pointer to the base of the user's UDL is returned in this register.

Code 6 — Unpack Binary Number

This call is used to translate a binary number contained in user General Register Zero into ASCII decimal or hexadecimal format The format of the parameter block is as follows.

PARBLK	ALIGN	2	
	DB	OPTION,6	DESIRED OPTION
	DC	A(DEST)	POINTER TO DESTINATION

The OPT field contains the supervisor call options. The ADDRESS OF DESTINATION field contains a pointer to a buffer in memory where the converted number is to be stored This buffer may begin on any byte boundary; it must be in a writable logical segment.

Options recognized are as follows

$X'00'+N$ Convert to hexadecimal
$X'40'+N$ Convert to hexadecimal, suppress leading zeros
$X'80'+N$ Convert to decimal
$X'C0'+N$ Convert to decimal, suppress leading zeros

N represents the length of the buffer, in bytes. If N is zero, a four-byte buffer is assumed N must be less than or equal to 63 $(X'3F')$ Note that 5 is sufficient for decimal and 4 for hexadecimal.

The converted number is right-justified in the buffer so that the least significant digit occupies the rightmost byte (highest address) in the buffer. If the number to be converted exceeds the buffer length, most significant bytes are lost. If suppression of leading zeros is requested, the number is stored in the buffer right-justified, and the remaining characters, if any, are filled with blanks.

The number to be converted must be supplied by the user in General Register Zero, and is assumed to be an unsigned 16-Bit binary number.

Code 7 — Log Message

This call provides access from the user task to the system console (or system log device) It gives the user a means of outputting a message with the assurance that it is printed on the console or log device regardless of Logical Unit assignments in force at the time of the call. A number of options are provided and the parameter block may take one of two forms, Direct and Indirect Text

Direct Text

PARBLK	ALIGN	2	
	DB	OPTION,7	DESIRED OPTION
	DC	H'LENGTH'	LENGTH OF MESSAGE IN BYTES
	DC	C'TEXT'	TEXT OF MESSAGE

Indirect Text

PARBLK	ALIGN	2	
	DB	OPTION,7	DESIRED OPTION
	DC	H'LENGTH'	LENGTH OF MESSAGE IN BYTES
	DC	A(TEXT)	ADDRESS OF MESSAGE TEXT

Options recognized are

$X'00'$ Direct text, Formatted message
$X'40'$ Indirect text. Formatted message
$X'80'$ Direct text, Image message
$X'C0'$ Indirect text, Image message

The LENGTH field specifies the length of the message in bytes. This message may be of any length and may begin on a byte boundary (for the Indirect text option).

The Formatted/Image option is similar to the ASCII formatted/Image option in SVC 1. Its effect on the system console or log device cannot be properly predicted by the caller, however, since there exists no mechanism for fetching the attributes of the system console or log device. The Command Processor, therefore, may restrict the Image option by executing it as thought it were Formatted for certain devices. The primary effect of the Image option on a TTY, Carousel, or CRT is to suppress the automatic carriage return at end of line, or to allow multiple line messages to be logged with one SVC call.

A Log Message call is treated as an I/O Proceed call. Message text is buffered within the system, enabling the user to modify or destroy the text or parameter block immediately following the call. If no entries are available on the log message queue, the message is lost. A message noting the lost message or messages is displayed in the system console.

Code 8 — Fetch Time

This call is used by a task to request the current time of day from the system. The system maintains a 24-hour clock calibrated in seconds since midnight. A value of zero indicates midnight; a value of 86399 (X'1517F') indicates 23·59:59. The clock may be interrogated in ASCII or binary format. The format of the parameter block is:

	ALIGN	2	
PARBLK	DB	OPTION,8	DESIRED OPTION
	DC	A(BUFFER)	ADDRESS OF BUFFER

Options recognized are:

X'00'   ASCII format
X'80'   Binary format

If ASCII is selected, the receiving buffer must be eight bytes long and may be aligned on a byte boundary, in a writable logical segment. The stored data is in the format:

      hh mm:ss

If Binary is selected, the receiving buffer must be four bytes long and aligned on a halfword boundary. The data stored in it is a binary fullword indicating seconds since midnight, as described previously.

Code 9 — Fetch Date

This call is used to request the present date from the system. The format of the parameter block is:

	ALIGN	2	
PARBLK	DB	0,9	NO OPTIONS
	DC	A(DEST)	ADDRESS OF RECEIVING BUFFER

The receiving buffer must be eight bytes long and aligned on a halfword boundary, in a writable logical segment. The system returns the date in the following format:

      mm/dd/yy

Alternatively, at system generation time, the following format may be selected:

      dd/mm/yy

This format is widely used outside the United States and in certain governmental (chiefly military) applications within the United States.

Code 15 — Pack Numeric Data

This call is the inverse of the SVC 2 code 6 call. It translates ASCII hexadecimal or decimal character strings to binary numbers. An option is provided to skip leading blanks. The format of the parameter block is as follows.

	ALIGN	2	
PARBLK	DB	OPTIONS,15	DESIRED OPTIONS
	DC	H'REG'	REGISTER HOLDING ADDRESS

Options recognized are:

X'00'	Hexadecimal
X'40'	Hexadecimal, skip leading blanks
X'80'	Decimal
X'C0'	Decimal, skip leading blanks

The REGISTER field of the parameter block specifies one of the user general registers. This register must contain a pointer to the first character of the ASCII string to be converted. The result is always returned in General Register Zero

This pointer register is returned pointing to the first byte in the string that could not be converted (If decimal is specified, this is the first non-numeric byte, if hexadecimal is specified, this is the first byte that is not 0-9 or A-F). If Register Zero is specified as REG. the updated input pointer is not returned.

The Condition Code (CC) is set to reflect the results of the processing:

CC=0 means a normal termination with no errors encountered in packing

CC=1 ('L' bit set) means no characters were processed, in this case, general register 0 is set to zero.

CC=4 ('V' bit set) means that the number processed was too large to be represented in 16 bits. If hexadecimal is specified, this means that more than four valid hexadecimal digits were processed. in this case, Register Zero contains the least significant four characters.

If decimal is specified, this means that the number processed is greater than $2^{16}-1$ (65535) In this case, Register Zero contains the number processed, modulo 2^{16}.

Code 16 — Pack File Descriptor

This call is used to convert an ASCII string containing a File Descriptor into the format necessary for a File Management Services Call (SVC 7). This call is particularly useful if the name of the system volume is not known to the caller The format of the parameter block is:

	ALIGN	2	
PARBLK	DB	OPTIONS,16	DESIRED OPTIONS
	DC	H'REG'	ASCII STRING ADDRESS REGISTER
	DC	A(DEST)	ADDRESS OF RECEIVING AREA

The REG field specifies one of the caller's general registers. This register must point to the ASCII string to be processed.

The RECEIVING AREA ADDRESS field points to a 16-byte receiving area. This receiving area must be aligned on a halfword boundary in a writable logical segment and is formatted as follows:

0 (0)		
	VOLUME NAME	
4 (4)		
	FILE NAME	
12 (C)		15(F)
	EXTENSION	(reserved)

Note that this is the same format as the File Descriptor field of an SVC 7 parameter block. The receiving area may in fact be such a field.

Options recognized are:

X'00'	Default System Volume
X'40'	Default System Volume, Skip Leading Blanks
X'80'	No Default Volume
X'C0'	No Default Volume, Skip Leading Blanks

If either "Skip Leading Blanks" option is selected, the SVC ignores all blanks from the current position of the pointer to the first non-blank. Otherwise, the File Descriptor to be converted is assumed to start at the current pointer position. The action of the "Default System Volume" options are shown later.

The pointer contained in Register REG is returned after the call pointing to the first byte that is not part of the File Descriptor. The Condition Code is set on return as follows:

0 normal
1 no volume name present in input
4 syntax error (input pointer not updated)
8 no extension present in input
9 no extension or volume name present in input

If a syntax error is detected, the contents of the receiving area are undefined. If volume name, file name or extension is fewer than 4, 8 or 3 characters long, respectively, the field is left justified and unused characters in the receiving area are blank-filled. The reserved character following the extension field is always set to a blank by the system.

If no volume name is provided and a "Default System Volume" option is selected, the current system volume name is moved into the VOLUME NAME field of the receiving area. If this option is not selected, the contents of the receiving area VOLUME NAME field are left unchanged.

For example:

	Input string	Receiving area	Condition Code
1.	DSC3:	DSC3bbbbbbbbbbbb	8 (no extension)
2.	ABC:FILE1.XY	ABCbFILE1bbbXYbb	0 (normal)
3.	DEF:FILE2	DEFbFILE2bbbbbbb	8 (no extension)
4	FILE4.PDQ	****FILE4bbbPDQb	1 (no volume)
5.	FILE5	****FILE5bbbbbbb	9 (no volume or extension)
6.	FRED:FILE5.	FREDFILE5bbbbbbb	0 (normal)
7.	$3%X,CP%	undefined	4 (syntax error)

**** = volume name field contents depend on selected option

Note that the selection of the blank extension (example 6) is not considered the same as the selection of no extension at all. If no extension at all is selected, it is assumed that the caller may wish to use some default value.

Code 17 – Mnemonic Table Scan

This call permits the user to decode command mnemonics in a way identical to the OS/16 MT2 Command Processor. The format of the parameter block is as follows:

	ALIGN	2	
PARBLK	DB	0,17	NO OPTIONS
	DB	REG1,REG2	INPUT, INDEX REGISTERS
	DC	A(MNEMONIC TABLE)	

The options byte must contain zero, as there are no options. The REG1 byte contains the number of a general register. This register is a pointer to the source string being scanned. The REG2 byte contains the number of a general register into which the result of the call is put. The MNEMONIC TABLE ADDRESS field contains the address of a mnemonic table within the user's program space

The format of a mnemonic table is a string of mnemonics. No boundary alignments are required. Mnemonics within the table are separated from each other by bytes containing X'00'. The end of the table is signified by the occurrence of two consecutive bytes of X'00'.

M N E M 1 00 M N E M 2 00 M N E M 3 00 00

The first byte of a mnemonic may be any ASCII character; subsequent bytes of the mnemonic must be alphabetic. A mnemonic may be of any length. Abbreviations are permitted in the same way as described in the operator command syntax (see *OS/16 MT2 Operator's Manual*). To indicate an abbreviation, required characters of a mnemonic are flagged with Bit 0 = 1, non-required characters are flagged with Bit 0 = 0 Required characters must be contiguous and must begin with the first character of a mnemonic. Thus, the mnemonic RECONFIGURE is coded as·

$$DB \quad C'R'+X'80', C'E'+X'80', C'C'+X'80', C'ONFIGURE',0$$

where the first three characters are flagged as required. Note the byte of zero at the end, this is the mnemonic terminator

The result, returned in Register REG2, is a number which is -1 if no match was found, or 0 to n-1, where n is the number of mnemonics in the table, if a match was found. This number represents the position of the matched mnemonic in the table, starting with zero. Thus, if a match is found on the third item in the table, the result returned in REG2 is 2.

REG1 is returned pointing to the first character that was not matched This is normally a separator following the mnemonic in the string being scanned. If no match is found, Register REG1 is returned unchanged.

If a match is found, the Condition Code is set to zero, if no match is found, the Condition Code 'V' bit is set (CC=4).

Code 18 — Move ASCII Characters

This call is used to move characters from an input string to a target string The number of characters moved may optionally be controlled by the presence of one or more terminating characters in the input string. The format of the parameter block is as follows:

	ALIGN	2	
PARBLK	DB	OPTION,18	OPTION
	DB	REG1,REG2	INPUT, OUTPUT POINTERS
	DC	A(ENSTRING)	ADDRESS OF ENDING CHARACTER STRING

REG1 specifies the register pointing to the input string and REG2 specifies the register pointing to the output string, which must be in a writable logical segment.

Options recognized are

$$X'00' + N \qquad \text{No ending characters}$$
$$X'80' + N \qquad \text{Use ending character string}$$

where N is the number of characters to be moved. N must be less than or equal to 127 (X'7F').

If option X'00' is selected, N characters are moved. The input and output string pointers are modified to point to the location following the last byte moved in the input and output areas, respectively. The Condition Code is set to zero.

If option X'80' is selected, each character moved is checked against the ending character string. If this character matches any of the ending characters, this character is not moved and the SVC terminates, modifying the pointers as described previously, and setting the Processor Condition Code to zero. If the Nth character is moved and no match has been found, the SVC terminates as described previously, but the 'V' bit of the Condition Code is set (CC=4) to signify that no ending character was found

The ending character string is formatted as follows·

M	Ending characters

where M is a byte containing the number of ending characters in the string. The length of the entire string, therefore, is M+1 bytes. A string containing, for example, the ending characters , , / is coded as follows

$$DB \qquad 5,C',.,;/'$$

Code 19 — Peek

This call permits the task to extract certain system and task-dependent information from system tables. This information is moved to the caller's parameter block, which is 26 bytes long. The format is illustrated in Figure 3-1.

0(0) 00	1(1) 19	2(2) NLU	3(3) MPRI
4(4) OSID			
12(C) TASK ID			
20(14) CTSW		22(16) TASK OPTIONS	
24(18) TASK STATUS			

Figure 3-1. SVC 2 Code 19 Parameter Block

The first two bytes of the parameter block are set by the user program and indicate SVC 2 code 19 The remaining fields are filled in by OS/16 and have the following meanings:

NLU	Number of LUs available to the task
MPRI	Maximum priority of the task
OSID	Eight ASCII characters of the form "OS/16MT2r" where r is numeric and indicates the revision level of the OS/16 MT2 system.
TASK ID	Eight-character ASCII Task Identifier.
CTSW	Status field (upper 16 bits) of the task's current Task Status Word.
TASK OPTIONS	The Options halfword from the task's TCB.
TASK STATUS	The status halfword from the task's TCB

This parameter must be on a halfword boundary in a writable segment and may be coded as follows:

PARBLK	ALIGN	2
	DB	0,19
	DS	24

NOTE

This parameter block must be 26 bytes long if information following it is not to be overwritten.

Code 23 — Timer Management

This call permits the user to set up a time interval, and either to wait for that interval, or to cause an item to be added to the task's Task Queue when the interval is complete. Through the trap capability, it provides a very useful mechanism for service of periodic functions. This call also permits the user to cancel an outstanding time interval.

The parameter block has the following format:

ALIGN	2	
DB	OPTIONS,23	DESIRED OPTIONS
DC	H'REG'	REGISTER CONTAINING PARAMETER
DC	Y'TIME'	TIME FIELD

Bits 0-3 (TT) of the TIME field indicate the type of interval desired The defined codes are:

0000	Seconds since midnight (Time of day interval)
0001	Milliseconds from now (Elapsed time interval)

All other codes are considered illegal, and cause an Illegal SVC error.

Bits 4-31 of the TIME field contain the number of clock ticks in the interval. This number can be no greater than 2^{28}-1, (268,435,455 decimal, 0FFFFFFF hexadecimal). A clock tick is a second if TT=0000 or a millisecond if TT=0001

Options recognized are as follows.

        X'00'   Add to Queue on completion of interval
        X'80'   Wait until completion of interval
        X'10'   Cancel time interval

If the Wait option is selected, the REG field of the parameter block is ignored The task enters a time Wait state and does not leave it until the completion of the interval.

If the Add to Queue option is selected, the REG field of the parameter block contains the number of one of the user's general registers. This register contains a parameter that is to be added to the user's Task Queue on completion of the interval A task may have only one active interval at any time. If a second interval is specified, it overrides the first. A time interval, once activated, cannot be deactivated unless the task goes to EOT before the interval is complete or a Cancel Time Interval is issued

If the Cancel Time Interval option is selected, the REG and TIME fields of the parameter block are ignored The outstanding time interval is cancelled and the Condition Code is set to zero. If no such interval is found, the V-bit of the Condition Code is set (CC=4).

SVC 3 – END OF TASK (EOT)

This call permits a task to terminate itself in an orderly fashion Its format is

        SVC 3,N   EOT

There is no parameter block associated with this call. Instead, the effective address of the second argument. N, is treated as a binary constant. and replaces the Return Code used by the Command Substitution System (CSS) (see the *OS/16 MT2 Operator's Manual)*

Return Codes may be treated as desired by the user in CSS conditional testing, however, the CSS system assumes that Return Code 0 represents normal termination, 255 represents termination by the system or another task (Cancel).

If the task issuing this call has I/O in progress at the time the call is made, the I/O is terminated Write operations are permitted to terminate normally, while Read operations are aborted.

If the task issuing this call is a nonresident task. its Logical Units are all closed, and it is removed from memory If, however. the task issuing this call is a resident task, its files are checkpointed, but not closed, and it is not removed from memory

N may consist of an indexed operand, e.g , 0(R1). In this case the Return Code is set to the sum of the contents of the specified register and the displacement.

SVC 5 – FETCH OVERLAY

This call permits a task to fetch an overlay from a specified Logical Unit The format of the parameter block is as follows.

PARBLK	ALIGN	2	
	DC	C'OVLYNM'	OVERLAY NAME (6 characters)
	DS	2	RESERVED FOR OS/32 COMPATIBILITY
	DB	0	STATUS
	DB	OPTIONS	OPTIONS
	DC	H'LU'	LU NUMBER

Options recognized are:

X'01'	Load from LU without positioning
X'04'	Load from LU after rewind

Any other value in the OPTIONS byte is considered illegal and results in an Illegal SVC error.

The status returned is:

X'00'	Overlay loaded successfully
X'10'	Load failed (for various possible reasons)
X'20'	Mismatch on overlay name
X'40'	Overlay would not fit in allocated memory

The overlay file must be assigned to the Logical Unit specified in the parameter block This file must be prepared by the OS/16 Task Establisher Task (TET/16). See Chapter 5 of the *OS/16 MT2 Operator's Manual* for the appropriate discussion of preparing tasks with overlays. The specified overlay name is checked against the name in the Loader Information Block of the overlay file. if it does not match, an error status is returned.

The calling program does not resume execution until the overlay is loaded. If the overlay is successfully loaded, the root program may Branch and Link to it as a subroutine. See Chapter 7 for a discussion of overlay facilities provided by TET/16.

If Load without positioning is specified, the file must be positioned at the LIB of the overlay to be loaded.

On return, the overlay file or device is positioned to the logical record following the last logical record containing the overlay.

The parameter block must be in a writable segment

SVC 6 – INTERTASK COORDINATION

SVC 6 provides facilities for foreground tasks to invoke and communicate with other tasks. SVC 6 is treated as an illegal SVC or a NOP (No Operation) from a background task, depending on the task option associated with the background task.

SVC 6 functions include the ability to:

Load a task
Start a task
Cancel a task
Queue a parameter to a task
Change a task's priority
Control Trap Generating Devices
Make a task memory resident or non-Memory Resident
Send a message to a task

The SVC 6 parameter block is 38 bytes long and must start on a halfword boundary in a writable segment. The format is illustrated in Figure 3-2.

```
┌─────────────────────────────────────────────┐
│ 0(0)                                          │
│                                               │
│                   TASK ID                     │
│                                               │
├─────────────────────────────────────────────┤
│ 8(8)                                          │
│                FUNCTION CODE                  │
├─────────────────────────────────────────────┤
│ 12(C)                                         │
│                 TASK STATUS                   │
├─────────────────────────────────────────────┤
│ 14(E)                                         │
│                 ERROR STATUS                  │
├──────────────────────┬──────────────────────┤
│ 16(10)               │ 17(11)               │
│        LOAD LU       │          PRIORITY    │
├──────────────────────┼──────────────────────┤
│ 18(12)               │ 19(13)               │
│          RPRI        │           RESERVED   │
├──────────────────────┴──────────────────────┤
│ 20(14)                                        │
│                START ADDRESS                  │
├─────────────────────────────────────────────┤
│ 22(18)                                        │
│  TT               DELAY TIME                  │
├─────────────────────────────────────────────┤
│ 26(1A)                                        │
│               DEVICE MNEMONIC                 │
├─────────────────────────────────────────────┤
│ 30(1E)                                        │
│                  PARAMETER                    │
├─────────────────────────────────────────────┤
│ 32(20)                                        │
│            MESSAGE BUFFER ADDRESS             │
├─────────────────────────────────────────────┤
│ 34(22)                                        │
│              RESERVED (4 BYTES)               │
└─────────────────────────────────────────────┘
```

Figure 3-2. SVC 6 Parameter Block

The meaning of each field is explained in the description of each function requiring that field Table 3-5 summarizes the use of each field. The parameter block may be coded:

	ALIGN	2	
PARBLK	DC	C'TASKID'	TASKID (8 bytes)
	DC	Y'function'	FUNCTION CODE
	DS	4	TASK ERROR STATUS
	DB	LU	LOAD LU
	DB	PRI	PRIORITY
	DS	2	RPRI/RESERVED
	DC	A(START)	START ADDRESS
	DC	Y'TIME'	TIME DELAY
	DC	C'DEVM'	DEVICE MNEMONIC
	DC	X'PARM'	PARAMETER
	DC	A(MESSAGE)	MESSAGE BUFFER ADDRESS
	DS	4	RESERVED

Although not all fields are used by each function, the full SVC 6 parameter block must be reserved.

TABLE 3-5. SVC 6 PARAMETER BLOCK FIELDS

FIELD	NAME	MEANING
Bytes		
0-7	Task ID	Name of task; not required if call is self-directed.
8-11	Function Code	Specifies desired functions, See Table 3-6
12-13	Task Status	The wait status halfword of the specified task is returned by the system in this field
14-15	Error Status	Set to ZERO for normal termination or to error code if error detected. See Table 3-7.
16	Load LU	Specifies LU from which to load the task. Used only for Load function.
17	Priority	Specifies priority for change priority function. May not be 255 (E-tasks). Must be in range 10-249 (U-tasks)
18	RPRI	Set by system to actual priority of specified task.
19		(reserved)
20-21	Start Address	Used only for START functions; specifies address at which to start specified task. If ZERO, signifies normal transfer address, as specified at TET/16 time.
22 (Bits 0-3)	TT	Time Type: indicates type of time delay Used only for Delay-Start. Legal codes are. 0000 = seconds since midnight 0001 = milliseconds from now.
22-25 (Bits 4-31)	Delay Time	Specifies number of seconds or milliseconds, as defined by TT field, to delay the start operation. Used only for Delay-Start.
26-29	Device Mnemonic	Specifies device affected by Connect, Thaw, SINT, Freeze and Unconnect functions. (See Table 3-6).
30-31	Parameter	Specifies parameter to be queued for Queue parameter function; specifies device parameter for Connect functions
32-33	A(Message)	Specifies address of message buffer for Send Message Function.
34-37		(Reserved)

Task ID and Function Code

The Task ID field specifies the name of the task at which the SVC 6 is directed. A Task ID must either be .BG for the background task or must consist of alphanumeric characters with the first character alphabetic. If the call is directed at the calling task (see function code D field), it is termed a self-directed call and the Task ID field is not required. A function may also be directed at the calling task by specifying Other-Task in the function code and the calling task's name in the Task ID field. The function code specifies the functions to be performed on the specified task. Each bit in the function specifies a separate function. The functions specified are performed in the order of designated bits from left to right. The definition of the function code bit assignments is given in Table 3-6.

Errors

If an error is detected in the performance of any requested function, an error status is returned and no further functions are performed. Since the Error Status field contains an indicator of which function was being performed at the time of error, it is always possible for the calling task to determine the functions that were performed and those that were not

The format of the Error Status halfword is as follows·

BIT	0 7	8 15
NAME	Position Pointer	Error Code

The Position Pointer points to the field in the Function Code fullword that was being processed at the time the error was detected. This pointer takes the form of a bit pointer which may have values in the range 0 through 31 The possible values of the Error Code field are detailed in Table 3-7.

Status Return

No matter what functions are specified, and except for Error Code 4 (no such task), the Wait Status halfword and the current priority from the specified task's TCB are always returned in the Task Status and RPRI fields. This allows the calling task to restore the specified task to its previous priority. If only a status return is desired, it is permissible to set no bits in the function code (except for the Direction field). The entire SVC 6 call is then treated as a Null operation, but the status and priority are returned as usual.

TABLE 3-6. SVC 6 FUNCTION CODES

0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	3	3
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
D		E				L		H			M	Q	P			O	T	I	F	U					N				A		
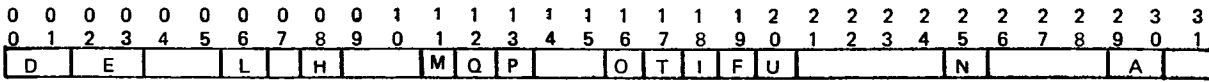
BIT(S)	NAME	HEX	MASK	MEANING
0-1	D	8000 C000	0000 0000	DIRECTION 00,01 = illegal codes 10 = Other Task 11 = self
2-3	E	0000 1000 2000 3000	0000 0000 0000 0000	END TASK. 00 = no function requested 01 = cancel 10 = delete 11 = delete
4-5				RESERVED (SET TO ZERO)
6	L	0200	0000	LOAD TASK
7				RESERVED (SET TO ZERO)
8	H	0080	0000	TASK RESIDENT
9-10				RESERVED (SET TO ZERO)
11	M	0010	0000	SEND MESSAGE
12	Q	0008	0000	ADD PARAMETER TO SPECIFIED TASK'S TASK QUEUE.
13	P	0004	0000	CHANGE PRIORITY OF SPECIFIED TASK.
14-15				RESERVED (SET TO ZERO)
16	O	0000	8000	CONNECT: CONNECT SPECIFIED DEVICE TO SPECIFIED TASK.
17	T	0000	4000	THAW· ENABLE INTERRUPTS ON SPECIFIED DEVICE.
18	I	0000	2000	SINT: SIMULATE INTERRUPT ON SPECIFIED DEVICE.
19	F	0000	1000	FREEZE: DISABLE INTERRUPTS ON SPECIFIED DEVICE.
20	U	0000	0800	UNCONNECT. DISCONNECT SPECIFIED DEVICE FROM SPECIFIED TASK.
21-24				RESERVED. (SET TO ZERO)
25	N	0000	0040	TASK NON-RESIDENT
26-28				RESERVED (SET TO ZERO)
29-30	A	0000 0000 0000 0000	0000 0002 0004 0006	START TASK: 00 = no function requested 01 = start immediately 10 = delay start 11 = delay start
31				RESERVED (SET TO ZERO)

TABLE 3-7. SVC 6 ERROR CODES

Code Dec (Hex)	Function(s)	Meaning
0(0)	All	No errors, all requested functions complete.
1(1)	All	Syntax error in Task ID field. Does not apply to self-directed calls
2(2)	--	Illegal function code
3(3)	L	Task already present.
4(4)	All but L	No such task in foreground.
5(5)	P	Invalid priority.
6(6)	L	Floating point not supported by SYSGEN.
7(7)	A	Specified task not dormant.
8(8)	S	Invalid Start address
10(A)	A (delay)	Invalid code in TT field
11(B)	M	Message not sent (Task unable to receive)
12(C)	Q	No queue, full queue, or entries disabled
13(D)	O,T,I,F,U	No such device in system.
14(E)	O,T,I,F,U	Device named is not a connectable device.
15(F)	O	Device is busy, cannot connect.
16(10)	T,I,F,U	Device not connected to specified task.
17(11)	L	Invalid or unassigned Load LU
18(12)	L	Checksum error.
25(19)	I	Device is not SINTable
66(42)	L	Resident Library or Task Common not present.
68(44)	L	Invalid format on Loader Information Block.
73(49)	L	Partition not vacant
74(4A)	L	I/O error on ROLL OUT
75(4B)	L	Assign/Close error on ROLL OUT.
76(4C)	L	Partition does not exist.
129-192 (81-C0)	L	I/O error reading Load LU, error status is as returned by SVC 1

End Task Function

Task ID and Function Code are the required fields.

This function causes the specified task to terminate as though it had executed an

SVC 3,255.

The Return Code of 255 indicates abnormal termination. If Delete is specified, the task is made nonresident and then cancelled. This causes it to be removed from memory. If Cancel is specified, the task is removed from memory only if it is a nonresident foreground task. If this call is self-directed, SVC 6 processing is terminated. It is not permissible to specify this function with a LOAD or START since this is a proceed function.

Load Task Function

Task ID, Function Code and Load LU are the required fields.

The specified task is loaded from the Load LU. If a task is already present in the system with the given Task ID or the Task ID is invalid, the call is rejected. A self-directed load request is also rejected

The task to be loaded must have been processed with the OS/16 Task Establisher Task (TET/16). The Loader Information Block (LIB) is read from the specified LU which must have been previously assigned and positioned to the LIB. The partition required by the task (determined at TET/16 time) is checked to see if it is vacant, or contains a rollable task of lower priority. If not, the call is rejected. Upon loading the task, it is given the name found in the TASKID field.

To load a task into the background, the TASKID must be .BG. The background partition must be dormant for the load to occur.

On successful return from this call, the specified file or device is positioned after the loaded task.

Send Message Function

Task Id, Function Code and Message Buffer Address are the required fields.

This function provides the calling task with the capability of sending a message to the called task. The task to which a message is directed must have been set up to receive messages. On request, a message set up in the calling task is accessed by the system as 64 bytes of data. The Task Id of the calling task is appended as a header and a 72 byte message is passed to the receiving task. If the receiving of messages by that task is disabled an error status is returned.

The Message Buffer must be on a halfword boundary. The actual message may be less than 64 bytes even though 64 bytes will be sent. No checking or formatting of the message is performed. Refer to the section further in this chapter entitled "The Sending and Receiving of Task Messages" for more information.

Queue Parameter Function

Task ID, Function Code and Parameter are the required fields.

The parameter specified in the PARAMETER field of the parameter block is added to the specified task's Task Queue, if:

- the specified task has a Task Queue;
- that queue is not full; and
- the specified task's TSW enables entries to the Queue.

Otherwise, the call is rejected with appropriate error status.

Change Priority Function

Task ID, Function Code and Priority are the required fields.

This function changes the priority of the specified task from whatever it was before the call to the priority specified in the PRIORITY field of the parameter block. The call is rejected for U-tasks if the priority specified is outside the valid range of 10-249.

If the specified priority is greater than the established maximum priority of the specified task then the established maximum priority is used instead of the specified priority. This is not considered an error. RPRI is set to the priority of the task before it is changed.

Task Resident and Non-Resident Functions

Task ID and Function Code are the required fields

The Task Resident function makes the specified task memory resident The Task Non-Resident function makes the specified task non-memory resident, allowing the task to be deleted when it goes to End of Task. Specifying Task Non-Resident and End Task does not delete the specified task because the task is cancelled and then is set Non-Resident.

Trap-Generating Device Functions

The functions Connect, Thaw, SINT, Freeze and Unconnect are provided for the control of Trap-Generating Devices (TGD) These functions all use the DEVICE MNEMONIC field of the parameter block This field must contain the mnemonic of a TGD. For all functions except Connect, a test is made to make sure that the specified TGD is connected to the specified task.

A TGD may not be connected to more than one task at a time; however, a task may have any number of TGDs connected to it. The parameter that is placed in the Task Queue on a TGD interrupt can be used to differentiate between multiple TGDs connected to a task

Connect

Task ID Function Code, Device Mnemonic and Parameter are the required fields.

This function connects the TGD specified by DEVICE MNEMONIC to the specified task. The named device must be a TGD, it must not be connected to any other task, or already connected to the same task. The parameter specified by PARAMETER becomes associated with the TGD; this parameter is placed in the specified task's Task Queue when an interrupt occurs. Interrupts are not enabled by this call; a Thaw call must be used to enable interrupts.

Thaw

Task ID, Function Code and Device Mnemonic are the required fields

This function enables interrupts on the specified TGD. The TGD is first checked to make sure that it is connected to the specified task Once enabled, interrupts on this device may be disabled only by a Freeze or Unconnect call, or by the task going to EOT. This call has no effect if interrupts are already Thawed.

SINT

Task ID, Function Code and Device Mnemonic are the required fields.

This function simulates an interrupt on the specified TGD. The device must be connected to the specified task and it must be a SINTable TGD. (Not all TGD drivers are capable of accepting a SINT call; see Chapter 8). If device interrupts have not been enabled by means of a Thaw call, the SINT call is ignored. This condition is not treated as an error.

Freeze

Task ID, Function Code and Device Mnemonic are the required fields.

This function disables interrupts on the specified TGD. It does not disconnect the device from the task. Note that interrupts are not queued while the TGD is in a frozen state This call has no effect if interrupts are already frozen.

Unconnect

Task ID, Function Code and Device Mnemonic are the required fields

This function detaches the specified TGD from the specified task. If the device is in a Thawed state, its interrupts are disabled. The device is now free to be Connected to any other task

Start Task Function

Task ID, Function Code and Start Address are the required fields. TT and Delay Time may optionally be specified.

This call causes the named task to be started. The address at which it is to be started is specified by the START ADDRESS parameter. If this field is non-zero, it indicates the address at which to start the task. If this field contains zero, the task is started at its established transfer address. If this field contains an address that is not within the specified task's memory, an error is returned by SVC 6.

This call is rejected if the specified task is not in a Dormant or Console Wait state. No implied-load facility is provided by this call; therefore, the specified task must be present in memory at the time of the call. It is permissible, of course, to specify both Load and Start functions in the same call, since the Start field is not processed until after the Load field processing is complete.

If Delay-Start is selected, the TT and DELAY TIME fields of the parameter block are used to determine the length and type of the delay. The specified task is actually started, but is placed immediately in a Time Wait or Interval Wait state.

This call is rejected if it is self-directed.

Sending and Receiving Task Messages

A task which is fully prepared to receive message(s) will have enabled queue entries on send requests from other tasks, and have set up a receiving buffer address in its UDL area. The task is not limited to one message buffer; it can set up a chain or ring of buffers so that a number of messages can be received.

Each receiving buffer must be 74 bytes long.

```
0        1 2         9 10                  73
┌──────────┬──────────┬────────────────────┐
│   LINK   │  TASKID  │      MESSAGE       │
└──────────┴──────────┴────────────────────┘
```
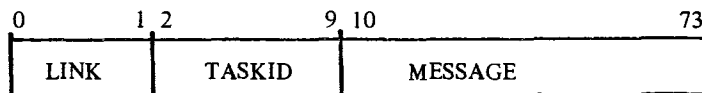
where, LINK is the link address (2 bytes) which points to next buffer or is zero to indicate last buffer in chain.

The least significant bit (i.e., X'0001') is used as a marker bit, since the true address is known to be even. This marker bit is:

1   Initialized zero by the task

2. Set by the system when a message is sent to indicate buffer full

3. Cleared by the task when the message is processed to indicate buffer empty

The TASKID field is an 8 byte Task Id of sending task and the MESSAGE field is a 64 byte message.

To receive messages a task must do the following.

1. Enable queue entries on send message by setting bit 11 in the current TSW. This enable bit is declared in the UDL STRUC (See Chapter 5) as:

    TSW.PMM   EQU   X'0010'   Queue Send Message

2. Set up a message buffer structure consisting of one or more 74 byte buffers. The address of the first buffer is declared in the UDL. This item is declared in the UDL STRUC as:

    UDL.MSGR   DS   2   A(Message Buffer)

    Link addresses must also be set up at the beginning of each buffer.

The task can take a trap whenever a message is received. Task Queue Service Traps are enabled by Setting bit 4 in the current TSW. This enable bit is declared in the UDL STRUC as:

    TSW.QSM   EQU   X'0800'   Queue Service Trap Enable

On a send message request, the system first verifies that the receiving task is in the system and checks its current Task Status Word (TSW) to determine whether queue entry on send message is enabled. If not, it returns an error status If enabled, the receiving buffer address is checked for being within the memory limits of the task. If zero (indicating no available buffer), or least significant bit is set (indicating the buffer is already full), or illegal, the system returns an error status

If a buffer is available, the system stores the Task Id of the sending task in bytes 3 to 10, gets the message start address from the SVC parameter block, and moves 64 characters into bytes 10 to 73. The marker bit in the link address field is set to indicate that the buffer is full. The address of the buffer is added to the task's Task Queue, and the link address is placed in the receiving task's UDL to point to the next available buffer.

Message Buffer Structures

The mechanism for passing messages gives the receiving task a choice of Message Buffer Structures. Figure 3-3 outlines four possible structures

Single-Buffer Ring

When the buffer is empty, the link field indicating the next message buffer points to itself, and the least significant marker bit is zero When a message is received, the link field is moved into the UDL as A(MESSAGE RING) and the marker bit is set to 1 This inhibits further messages since the next buffer is already full. When the task has processed the message, it clears the marker bit to again enable the receiving of a message into the same buffer.

Single-Buffer Chain

Here the link field is zero to indicate no other buffers are available for the receiving of messages On receipt of the first message, the zero link field is moved into the UDL, and will inhibit all subsequent messages To enable messages, the receiving task must dynamically set up a new empty buffer address in the UDL.

Mutliple-Buffer Ring

Initially all link fields indicate an empty buffer When buffer 1 is filled, A(MESSAGE RING) is set to a point to buffer 2. When buffer 2 is filled, A(MESSAGE RING) is set to a point to buffer 1. If, by the time the next message is ready to be sent, buffer 1 has been processed by the receiving task, and the receiving task has set buffer 1 link field "empty", the message is placed in buffer 1. As long as the receiving task can process buffers as fast as they are being filled, no messages are lost and no "send message" requests are rejected. A multiple-buffer ring may consist of as many buffers as the user task desires. The link field of each buffer is set to point to the next buffer, that of the last buffer is set to point to the first.

Multiple-Buffer Chain

This structure is similar to that of the multiple-buffer ring, with the exception that the link field of the last buffer is set to zero. When the last buffer is filled by the system, A(MESSAGE RING) is set to zero, inhibiting the sending of further messages This structure is of value if the receiving task is dynamically acquiring buffers from a pool within itself. Processed buffers are released from the chain and new buffers added to its end. If new buffers can be acquired as fast as the system fills them. no messages are lost.

A (MESSAGE RING)
IN UDL
BUFFER

A(BUFFER)

A(BUFFER)

DATA AREA

(a) Single-Buffer Ring

A (MESSAGE RING)
IN UDL
BUFFER

A(BUFFER)

0

DATA AREA

(b) Single-Buffer Chain

A (MESSAGE RING)
UDL
BUFFER 1

A(BUFFER 1)

A(BUFFER 2)

BUFFER 2

A(BUFFER 1)

DATA AREA

DATA AREA

(c) Multiple-Buffer Ring

A(MESSAGE RING)
IN UDL
BUFFER 1

A(BUFFER 1)

A(BUFFER 2)

BUFFER 2

0

DATA AREA

DATA AREA

(d) Multiple-Buffer Chain
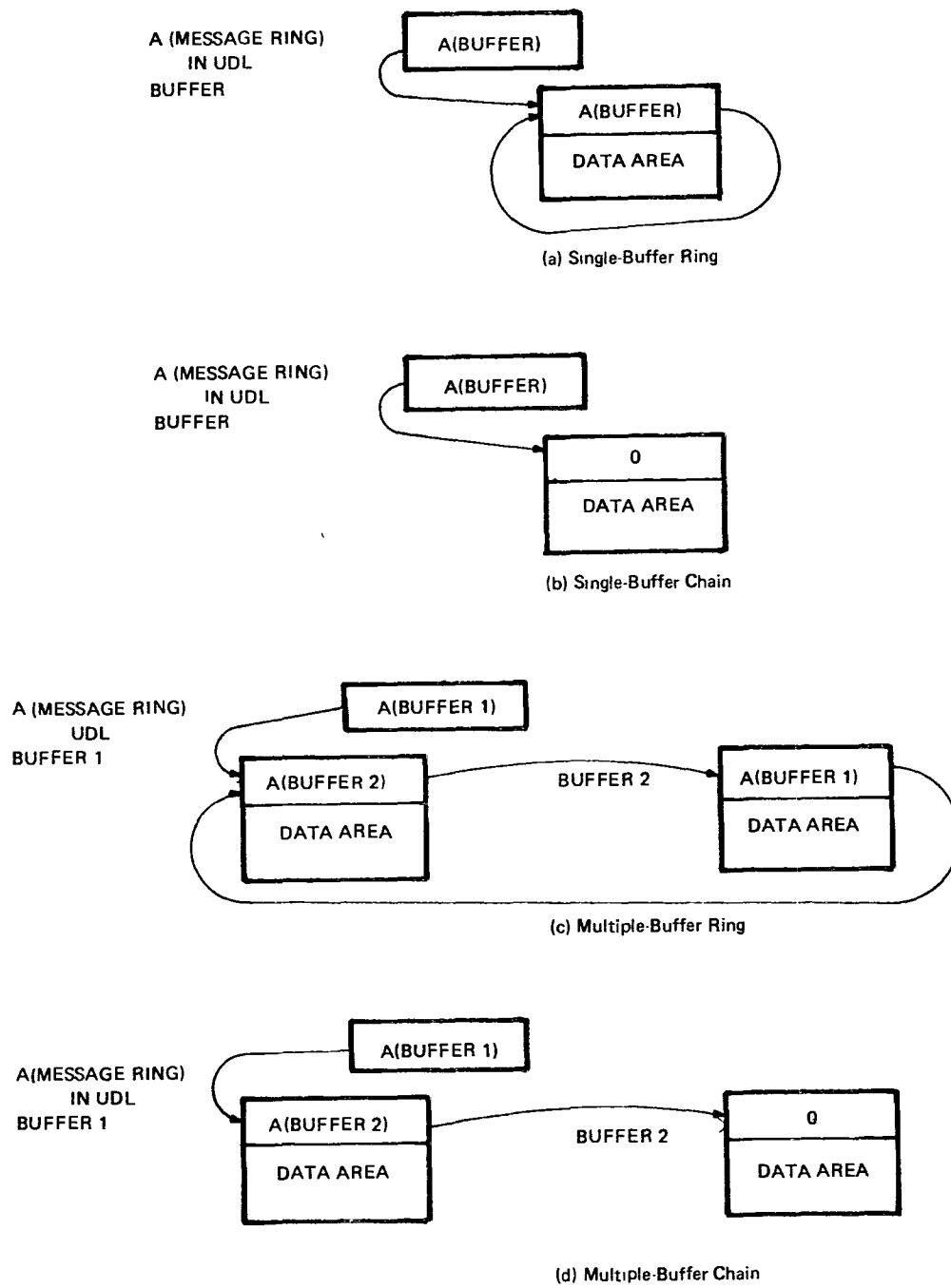
Figure 3-3. Message Buffer Structures

SVC 7 – FILE MANAGEMENT SERVICES

This call gives the user facilities for manipulation of files and devices. The facilities provided are:

  Allocation (creation) of direct-access files;
  Assignment of files and devices to Logical Units (LUs);
  Modification of access privileges on existing assignments;
  Renaming of files;
  Modification of files' protection keys;
  Closing (deassignment) of assigned files and devices;
  Deletion of direct-access files;
  Checkpointing of assigned files,
  Examination of attributes of assigned files and devices

The format of the SVC 7 Parameter Block, which must be on a halfword boundary in a writable logical segment is illustrated in Figure 3-4.

0(0) CMD	1(1) MOD	2(2) STAT	3(3) LU
4(4) WKEY	5(5) RKEY	6(6) LOGICAL RECORD LENGTH (LRECL)	
8(8) VOLUME ID OR DEVICE MNEMONIC			
12(C) FILENAME			
20(14) EXTENSION			23(17) (RESERVED)
24(18) SIZE			

Figure 3-4. SVC 7 Parameter Block

The meaning and use of each field is explained in the description of each function requiring that field. The parameter block may be coded

		ALIGN	2	
PARBLK	DB	CMD	COMMAND
	DB	MOD	MODIFIER
	DB	0	STATUS
	DB	LU	LU
	DB	WKEY	WRITE KEY
	DB	RKEY	READ KEY
	DC	H'LRECL'	LOGICAL RECORD LENGTH
	DC	C'VOLN'	VOLUME
	DC	C'FILENAME'	FILE NAME
	DC	C'EXT'	EXTENSION
	DC	Y'SIZE'	SIZE

SVC 7 Parameter Block Fields

Command/Modifier

The format of the Command/Modifier Halfword is shown in Figure 3-5

3-28

BIT	0	1	2	3	4	5	6	7	8	10	11	12	13	15
NAME	A	O	H	N	P	C	D	T	AP		BM		FT	

COMMANDS			MODIFIERS
Bits	Name	Hex Mask	Meaning
0	A	8000	Allocate - requires file type (FT) modifier
1	O	4000	Assign - requires access privilege (AP) and buffer management (BM) modifiers
2	H	2000	Change Access Privilege - requires access privilege (AP) modifiers
3	N	1000	Rename - no modifiers
4	P	0800	Reprotect - no modifiers
5	C	0400	Close - no modifiers
6	D	0200	Delete - no modifiers
7	T	0100	Checkpoint - no modifiers
			Bits 0-7 all zero = Fetch Attributes (F)
8-10	AP		Access Privileges 0000
		0000	SRO - Shared Read Only
		0020	ERO - Exclusive Read Only
		0040	SWO - Shared Write Only
		0060	EWO - Exclusive Write Only
		0080	SRW - Shared Read-Write
		00A0	SREW - Shared Read-Exclusive Write
		00C0	ERSW - Exclusive Read-Shared Write
		00E0	ERW - Exclusive Read-Write
11-12	BM		Buffer Management
		0000	Default Buffer Management
		0008	Unbuffered (Physical)
		0010	Buffered (Logical)
		0018	Reserved
13-15	FT		File Type
		0000	Contiguous
		0001	Reserved (considered illegal)
		0002	Indexed
		0003	
		....	Reserved (considered illegal)
		0007	

Figure 3-5. SVC 7 Command/Modifier Halfword

Command

The Command bits are processed sequentially from left to right. If Bits 0-7 are all ZERO, a Fetch Attributes function is specified. The specific functions are described in the following paragraphs.

Access Privileges

Access privileges are necessary for Assign and Change Access Privilege calls. This field is ignored for all other functions If Bits 8-10 are all ZERO, Shared Read Only (SRO) is specified.

Buffer Management

The default buffer management methods are Unbuffered for Contiguous File Type and Buffered for Indexed File Type In OS/16 MT2, this field is ignored and the default is always used. For a Fetch Attributes call, the device code is returned to the Modifier byte of the Command/Modifier halfword.

Status

The interpretation of the status field depends on the functions specified in the call and is defined under the description of each command. A status of zero always means the desired options were performed without error. A summary of all possible error codes is given in Table 3-8 for reference.

TABLE 3-8. SVC 7 STATUS BYTE

ERROR CODE DEC (HEX)	FUNCTION	MEANING
0 (00)	ALL	No error, the requested functions are complete.
1 (01)	A,O,D	Illegal function, Illegal file type
2 (02)	All but A,D	LU error; illegal LU
3 (03)	A,O,D	Volume error; no such volume/device in system
4 (04)	A,O,N.D	Name error; mismatch on filename or extension field.
5 (05)	A,O	Size error; erroneous LRECL or size field or no room on disc.
6 (06)	O,P,D	Protect error; erroneous protection keys.
7 (07)	All but C,T,F	Privilege error; unable to obtain requested privilege.
8 (08)	O	Buffer error; no room in system for file control blocks or buffers.
9 (09)	All but A	Assignment error, LU not assigned.
10 (0A)	A,O,N.P,D	Type error, non-direct access device, or device off-line.
11 (0B)	A,O,N.D	File Descriptor error; Illegal syntax.
12 (0C)	O	Attempt to assign SVC 6 connectable device.
129-192 (81-C0)	All but F	I/O error; interpreted as SVC 1 status byte

The first error detected causes the SVC to return. If multiple functions were specified (e.g., Allocate and Assign) some functions may have been properly performed (always in left-to-right sequence).

LU Field

This byte defines the Logical Unit used for all the SVC 7 functions except Allocate and Delete

Write Key And Read Key

Protection keys for devices and Direct-Access files are specified in this halfword. These keys are required for the Allocate, Assign, Reprotect, and Delete functions. This field is used by the Fetch Attributes call to return the device or file attributes.

Logical Record Length (LRECL)

This halfword field, on an Allocate call, must contain the logical record length for a buffered file. On a Fetch Attributes call, the logical record length of a file or physical record length of a device is returned in this field If ZERO is returned, device or file has a variable record length. LRECL is not used for other functions.

Volume ID (VOLN) or Device Mnemonic

This four-byte field identifies the volume if it is a Direct-Access file, or the device mnemonic if it is a device. This field of ASCII characters is required for the Allocate, Assign, and Delete.

VOLN together with FILENAME and EXT fields specify a File Descriptor. The volume name or device name, as the case may be, is returned by the system on a Fetch Attributes call in the VOLN field.

Filename

This eight-byte field identifies the file on a Direct-Access device; it is ignored for a device. The user must specify the filename in ASCII characters for Allocate, Assign, Rename, and Delete calls. The filename is returned by the system on a Fetch Attributes call; it is blank for a device.

Extension

This three-byte ASCII field identifies the file type (e.g., OBJ, TSK, CSS, etc.), on a Direct-Access file. It is treated as an extension of, and is required under the same conditions as, the FILENAME. The byte following this field is reserved for future expansion and is ignored.

Size

This fullword field, on the Allocate call, must contain the file size in sectors for a Contiguous file, or the data block size (Bits 0-15) and index block size (Bits 16-31), in sectors for an Indexed file. On a Fetch Attributes call, this field is used to return the current size of a file, or buffer size in bytes for a communications terminal; size is unchanged for a device For an Indexed file, size is in number of logical records; for contiguous files, it is in number of sectors. For a Direct-access device (E-Task only) this field returns the number of sectors on the device.

SVC 7 Functions

Allocate

The Allocate function reserves space on a Direct-Access device and in the directory for the specified file type (FT). In the case of Indexed files, only a directory entry is reserved. The protection keys are entered in the directory. The required parameters are FT, KEYS, LRECL, VOLN, FILENAME, EXT and SIZE.

Applicable error codes are

1(01)	Illegal Function; an illegal file type was specified
3(03)	Volume Error, the specified volume was not mounted
4(04)	Name Error, the specified file name already exists on the specified volume
5(05)	Size Error; if a Contiguous file, there is not sufficient contiguous space on the specified volume to allocate a file of the specified size; if an Indexed file, there is insufficient space for a directory entry or invalid block size or logical record length specified.
7(07)	Privilege error; Disc is Write protected.
10(0A)	Type Error, the specified volume is not a Direct-Access volume.
11(0B)	File Descriptor Error; invalid syntax in volume name, file name or extension fields.
129-192 (81-C0)	I/O error

Assign

The Assign function establishes a logical connection between a file (or device) and the task through a specified Logical Unit, under a given access privilege (AP) and using a given Buffer Management (BM) technique. The call is processed as follows· First the access privilege is examined to determine which protect key to check. The proper key is then checked against the keys in the file directory. The BM field is checked to see if the specified Buffer Management technique is valid for the type of file being assigned. The file is then assigned according to the requested access privileges. If SWO or EWO (Write Only) is specified, the file is positioned at its logical end (records are appended). Otherwise, the file is positioned at the beginning (record number = zero). For devices, only the keys are checked, for the given access privilege.

The required parameters are AP, BM, LU, KEYS, VOLN, FILENAME and EXT; BM, FILENAME and EXT fields are not used for devices.

Applicable error codes are.

1(01)	Illegal function; attempt to assign invalid file type.
2(02)	LU Error; Illegal LU
3(03)	Volume Error: no such volume, Direct-Access device specified for U-Task
4(04)	Name Error; no such name on given volume
5(05)	Size error; no room on disc to allocate an index and a data block
6(06)	Protect Error; mismatch on protection keys
7(07)	Privilege Error, requested privilege may not be granted
8(08)	Buffer Error, no room for FCB or buffer
9(09)	LU Already Assigned
10(0A)	Type Error; attempt to assign an off-line device to a U-Task.
11(0B)	File Descriptor syntax error.
12(0C)	Attempt to assign a Trap Generating Device (TGD)
129-192 (81-C0)	I/O Error

NOTE

A buffered file requires a buffer equal to the sum of the index and data block size of the file to be allocated in system space. The BM field is ignored in release 00 and 01; the default buffer management technique is used.

Change Access Privileges

This function allows the user to change the current access privileges of a file or device which is assigned Only two parameter fields in the SVC 7 Parameter Block are required, the LU and AP portion of the modifier field.

If an error is encountered while processing this request, the file remains assigned with its original access privilege.

Applicable error codes are·

2(02)	LU Error, illegal LU
7(07)	Privilege Error; new privileges cannot be granted
9(09)	Assignment Error; LU not assigned
129-192 (81-C0)	I/O Error

Rename

This function permits the name of an assigned file to be changed. The file must currently be assigned to the specified Logical Unit for Exclusive Read-Write (ERW). The required parameters are LU, FILENAME and EXT. The given LU must be assigned to a direct-access file (unless the caller is an Executive Task which may rename devices) The volume name field of the parameter block is ignored. The specified FILENAME EXT replaces the previous FILENAME.EXT in the directory if the Rename function is successful.

Applicable error codes are·

2(02)	LU Error, illegal LU
4(04)	Name Error; new name already exists on given volume
7(07)	Privilege Error; file not assigned for ERW
9(09)	Assignment Error; LU not assigned
10(0A)	Type Error, LU assigned to a device (U-Task) or to the Null device (E-Task)
11(0B)	File Descriptor Error
129-192(81-C0)	I/O Error

Reprotect

This function permits the protection keys of an assigned file to be changed. The required parameters are KEYS and LU. The given LU must be assigned to a Direct-Access file for Exclusive Read-Write (ERW) (unless the caller is an E-Task which may modify the protection keys of devices). If either the specified Read key or Write key is equal to X'FF', that key is ignored rather than changed (unless the caller is an E-Task). If the call is rejected, the previous keys remain unchanged.

Applicable error codes are:

2(02)	LU Error; illegal LU
6(06)	Protect Error, current key has value X'FF' (U-Task only)
7(07)	Privilege Error; not assigned for ERW
9(09)	Assignment Error; LU not assigned
10(0A)	Type Error; LU assigned to a device (U-Task) or to the Null or console device (E-Task)
129-192(81-C0)	I/O Error

Close

This function discontinues an assigned logical connection between a task and a file or device. LU is the only required parameter. The specified LU is deassigned Files or buffered communications devices assigned for Write access have any partially filled buffers, written out by the Close call. The Close function issues an SVC 1 Wait Only call to ensure that any previous I/O Proceed calls are complete before deassigning.

Applicable error codes are:

2(02)	LU Error; Illegal Logical Unit
9(09)	Assignment Error; LU not assigned
129-192(81-C0)	I/O Error

Delete

For a Delete function, the VOLN, FILENAME, EXT, and KEYS parameters must specify a Direct-Access file which is not currently assigned. If these conditions are met and both Read and Write keys match, the file is deleted from the directory of its volume. Only an E-Task may delete a file with a key of X'FF'.

Applicable error codes are:

1(01)	Illegal Function; attempt to delete invalid file type
3(03)	Volume Error; no such volume
4(04)	Name Error; file name does not exist on volume
6(06)	Protect Error; invalid protection keys
7(07)	Privilege Error; device write protected
9(09)	Assign Error; file currently assigned.
10(0A)	Type Error; attempt to delete a device.
11(0B)	File Descriptor Error
129-192(81-C0)	I/O Error

Checkpoint

The Checkpoint function permits the user to flush system Buffer Management buffers and to update the directory entry for a buffered file or communications device (e.g , BISYNC) on a given LU LU is the only required parameter. Requesting a checkpoint for a non-buffered file or device has the same effect as an SVC 1 Wait Only call.

The user may wish to employ Checkpointing after sensitive data is added to a buffered file because the logical blocking of data in memory in system buffers leaves the file vulnerable. The integrity of the data can be preserved on the file by Checkpointing. In case of system failure, all data on Indexed files up to the latest Close or Checkpoint operation is recoverable; data appended after the most recent Checkpoint is lost. Checkpoint differs from a Close/Assign sequence in that no repositioning is performed and that filename, access privileges, and keys need not be specified.

The applicable error codes are:

2(02)	LU Error; illegal Logical Unit
9(09)	Assignment Error; LU not assigned
129-192(81-C0)	I/O Error

Fetch Attributes

Certain programs may require, for proper operation, knowledge of the physical attributes of the device or file associated with a given LU. For example, it might be desirable to know if random access is possible or if the device is rewindable. The Fetch Attributes function gives the user access to this information.

The applicable error codes are:

2(02)	LU Error; Illegal LU
9(09)	Assignment Error; LU Not Assigned

Various fields within the SVC 7 parameter block are redefined for this call. When the command field is set to zero, indicating a Fetch Attributes call, the only required parameter is the Logical Unit. The system returns information in fields KEYS, LRECL, VOLN, FILENAME, EXT, SIZE and Modifier byte.

The Write Key/Read Key halfword is redefined to receive an Attributes halfword as given in Table 3-9. Any bit set means the device or file supports the corresponding attributes.

TABLE 3-9. SVC 7 DEVICE ATTRIBUTES HALFWORD

BIT	ATTRIBUTES
0	Interactive device
1	Supports Read
2	Supports Write
3	Supports Binary
4	Supports Wait I/O
5	Supports Random
6	Supports Unconditional Proceed
7	Supports Image
8	Supports Halt I/O
9	Supports Rewind
10	Supports Backspace Record
11	Supports Forward Space Record
12	Supports Write Filemark
13	Supports Forward Space Filemark
14	Supports Backspace Filemark
15	Reserved

The LRECL field is set by the system to the physical record length associated with the device (e.g., 80 for a card reader, 120 or 132 for a line printer) if the record length is fixed; these two bytes are set to zero for a variable-record length device, (e.g., magnetic tape). A TTY or CRT, which is in the strict sense a variable-record length device, normally has LRECL set as though it were a fixed-record length device This is because such a device is normally used in a fixed-record length method. Furthermore, a Contiguous Direct-Access file is considered to have a variable record length, whereas an Indexed file is considered to have a fixed record length, which is the logical record length specified for that file at the time of its allocation.

The volume name, file name, and extension (VOLN.FILENAME.EXT) for a named file, or the device mnemonic for a device is returned in the File Descriptor portion of the parameter block. The program may use this information to provide the operator with intelligent information in case of error, to generate the name of a new output file by transformation on the name of the input file, etc.

The current size of a Direct-Access file or current buffer size of a buffered communication device is returned in the SIZE field; SIZE is unchanged for devices. The number of logical records is returned for an Indexed file; the number of sectors is returned for a Contiguous file. These sizes are returned as unsigned hexadecimal numbers. The extended device code for a communications device is returned in the high order halfword of the SIZE field.

The Modifiers byte is set to indicate the file or device type. A sample of device codes is given in Table 3-10. The codes for all supported devices are given in Chapter 8.

Note that the full parameter block must be specified for the Fetch Attributes call or the system may overwrite the information following the parameter block.

TABLE 3-10. EXAMPLE DEVICE CODES

CODE DEC (HEX)	FILE OR DEVICE TYPE
0(0)	CONTIGUOUS FILE
2(2)	INDEXED FILE
16(10)	MODEL 33 TTY KEYBOARD/PRINTER
17(11)	MODEL 35 TTY KEYBOARD/PRINTER
18(12)	NONEDITING CRT ON CURRENT LOOP INTERFACE
34(22)	NONEDITING CRT ON LOCAL PASLA
36(24)	GRAPHIC DISPLAY TERMINAL ON LOCAL PASLA
49(31)	2.5 MBYTE REMOVABLE DISC CARTRIDGE
50(32)	10 MBYTE FIXED DISC CARTRIDGE
51(33)	10 MBYTE REMOVABLE DISC CARTRIDGE
52(34)	40 MBYTE REMOVABLE DISC CARTRIDGE
64(40)	800 BPI MAGNETIC TAPE
65(41)	1600 BPI MAGNETIC TAPE
66(42)	INTERTAPE CASSETTE
80(50)	HIGH SPEED PAPER TAPE READER/PUNCH
81(51)	MODEL 33 TTY READER/PUNCH
82(52)	MODEL 35 TTY READER/PUNCH
96(60)	CARD READER WITHOUT HOLLERITH/ASCII TRANSLATION
97(61)	CARD READER WITH HOLLERITH/ASCII TRANSLATION
112(70)	LOW SPEED LINE PRINTER
114(72)	HIGH SPEED LINE PRINTER
255(FF)	NULL DEVICE

SVC 9 – LOAD TSW

This call is used to return from task-handled traps, it is also used to change trap enable/disable bits, to change queue entry enable/disable bits, and to enter Trap Wait. This is the task analogue of the machine-level LPSW instruction.

The format of this call is·

    SVC 9, A(X2)

The effective address of the SVC instruction specifies the location at which the new TSW is to be found on a halfword boundary in the task's address space.

The effect of this instruction is to replace the Current Task Status Word found in the task's TCB with the indicated Task Status Word. Unless Trap Wait is specified in the new TSW, the task resumes executing instructions at the address specified by the LOC field of the new TSW.

If the LOC field of the new TSW is zero, execution resumes at the instruction following the SVC 9.

CHAPTER 4
GUIDE TO USING SVC 2 FACILITIES

COMMAND PROCESSORS

Command processors are a part of almost all utility programs and of many applications programs. Every time a program is written that has to accept directions from the outside, either by conversation with an operator or by job-control statements read from an input device, a command processor is required to interpret those directions. Consequently, command processors are constantly being written and rewritten, tested and retested, yet most command processors are so similar to each other that one could generally be put together out of a package of "canned" routines, if such a package were at hand.

Some high-level languages, for example, SNOBOL, and certain versions of Extended BASIC, contain string-handling routines of sufficient sophistication as to render the writing of a command processor trivial. However, the assembly language programmer has no such facilities.

One of the earliest functions of operating systems was to simplify the writing of I/O handling code. A logical extension to this concept is the provision, within the operating system, of a simple and well-defined method of using the operating system's own command processing routines. The INTERDATA OS/16 MT2 Operating System provides such a method.

The user of this method may find that the command language of a program is constrained to the use of certain formats if the operating system's command processor calls are chosen. These calls provide such a great range of capabilities, and are capable of such flexible use, as to make a slight restriction in formatting negligible in cost when compared to the savings in design, code and debug time that are realized by the use of these calls.

This chapter gives the user a guide to the use of the calls provided in OS/16 for command processing functions.

COMMAND STATEMENT INPUT

The Command Statement may be read via SVC 1 from any Logical Unit, or may be passed from one task to another. The important issue is how the Command Statement is handled once it is in the buffer.

First, the buffer address is put into a given register. This register is used as the current buffer pointer during the entire command processing operation. All four command processing SVC 2 calls (Pack, Pack File Descriptor, Mnemonic Table Scan and Move ASCII Characters) expect to find the current buffer pointer in a register. The choice of which register to use is made by the user.

Mnemonic Scan

A Command Statement usually starts with some kind of mnemonic. This first mnemonic is generally called a *verb* because it tells what to do.

OS/16 provides a call, Mnemonic Table Scan, which may be used to look for a match on any table of mnemonics. These mnemonics must be all alphabetic characters, except for the first character, which may be any ASCII character. Thus, $FRED is a valid mnemonic, but GLA2H is not. The mnemonic table is organized as a byte string. Each mnemonic in the table must be separated by one Null character (X'00') from the following mnemonic. The table is terminated by two Null characters in a row.

The OS/16 mnemonic syntax permits abbreviations. These are represented in the mnemonic table by flagging all required bytes of the mnemonic with Bit 0 set to a 1 Non-required bytes have Bit 0 set to 0. To represent the mnemonic APPEND, the code within the mnemonic table is:

```
        DB    C'A'+X'80', C'P'+X'80'        required characters
        DB    C'PEND'                       non-required characters
        DB    0                             end of mnemonic
```

In order to match a mnemonic table entry, the command string must match all required characters. If more alphabetics are present in the command string, they must match the non-required characters. If all required and non-required characters are exhausted and there are still more alphabetics in the command string, no match is recognized.

For the example given previously, a legitimate match is found on

        AP
        APP
        APPE
        APPEN
        APPEND
        AP2   (scan terminates on first non-alphabetic after first character)

No match is found on·

        A   (too short)
        APPENG   (no match on non-required characters)
        APPENDIX   (too long)

The mnemonic table scan routine terminates as soon as it finds a non-alphabetic character in the command string If a valid match is found at this point, an index number indicating the position within the mnemonic table of the matched mnemonic is returned. If no match is found, the index number is returned set to -1. The system does not check the separator (the non-alphabetic character that terminated the scan routine). This character must be checked by the user. The buffer pointer that was given to the system is returned pointing to the separator, if a match was found; it is returned unchanged if no match was found. Entries in the mnemonic table are counted from ZERO, that is, if a match was found on the third mnemonic in the table, the index is returned with a value of 2.

The condition code 'V' bit is also set to show whether or not a match was found It is set to 0 if there was a match, or to 1 (CC=4) if there was no match.

A sample of how to use the Mnemonic Table Scan routine follows Assume that the mnemonic table is to consist of four entries: SORT, MERGE, PRINT, and STOP. This table is coded as follows:

        MNMTAB   DB   C'S'+X'80', C'O'+X'80',C'RT',0
                  DB   C'M'+X'80',C'E'+X'80',C'RGE',0
                  DB   C'P'+X'80',C'R'+X'80',C'INT',0
                  DB   C'S'+X'80',C'T'+X'80',C'OP',0,0

Notice the two bytes of ZERO ending the mnemonic table. If the command buffer pointer is found in register PNTR and the index is to be returned in register INDX, an SVC 2 parameter block to handle this table is coded as follows:

        SCAN   DB   0,17,PNTR,INDX
                  DC   A(MNMTAB)

Assume that a command line has just been read into a buffer named CMDBUF. The verb is found as follows.

        FNDVB   LDAI   PNTR,CMDBUF   set up buffer pointer
                  SVC   2,SCAN   scan the table
                  BO   CMDERR   branch if no match
                  SLLS   INDX,LADC   prepare for indexed branch
                  LDA   INDX,JTAB(INDX)   get jump table address
                  BR   INDX   go to command executor

        JTAB   DAC   SORT, MERGE, PRINT, STOP

This routine did not check the separator; if the characters SORT* had been in the command buffer, the result would have been the same as if SORT (Carriage Return) had been the command buffer content. Using this information, the code can be modified by taking advantage of the fact that PNTR points to the separator after a valid match. Assume that only a Carriage Return is allowed to follow the verb:

        FINDVB   LDAI   PNTR,CMDBUF
                  SVC   2,SCAN
                  SLLS   INDX,LADC
                  BM   CMDERR   another way to test for error
                  LIS   WORK, X'0D'   pick up a carriage return
                  CLB   WORK,0(PNTR)   check the separator
                  BNE   CMDERR
                  LDA   INDX,JTAB(INDX)
                  BR   INDX

The results are better. If different separators are required for different verbs, however, the separator check should be done after the jump table branch. If no match is found, PNTR is not changed. If the proper mnemonic is not found, consult a different mnemonic table. The verb, in this particular syntax, may be optional and therefore has a default value. For example, in BASIC the verb is optional and defaults to LET.

Operand Decoding

Most command languages require operands for most of their verbs. The verb tells the processor what to do, the operands tell what to do it to, or how to do it. Operands in OS/16 MT2 command processor syntax are

        Decimal numbers up to 65,535
        Hexadecimal numbers up to FFFF
        File Descriptors
        Mnemonics
        Arbitrary ASCII strings

These are sufficient for most needs. If other forms of modifiers are required, they may be decoded by the user.

Decimal and Hexadecimal Numbers

The OS/16 MT2 Pack Numeric Data call allows decimal and hexadecimal numbers to be decoded. Options are provided either to skip, or not to skip, leading blanks. The current buffer pointer is required; this may be in the same register that was used for the MNEMONIC TABLE SCAN call. The result is always returned in Register 0. This call sets the Condition Code to indicate the status of the call. If too many digits were found, the 'V' flag is set (CC=4); if no digits were found, the L flag is set (CC=1). The first nontranslatable character found terminates the operation, and PNTR is set to point to that character, so that the separator can be checked easily.

The parameter block looks like one of the following (depending on options)

PACKDEC	DB	X'80',15,0,PNTR	decimal
PACKHEX	DB	0,15,0,PNTR	hexadecimal
PACKDECS	DB	X'C0',15,0,PNTR	decimal, skip leading blanks
PACKHEXS	DB	X'40',15,0,PNTR	hexadecimal, skip leading blanks

Assume that the syntax of the rewind command were as follows

        <rwdcmd> .:= REWIND <lu>

        <lu> ::= <decimal number less than 256>

And assume that the verb finder had found a match on REWIND and now wishes to find <lu>. This is done as follows.

REWIND	SVC	2,PACKDECS	get decimal number, skip blanks
	BO	CMDERR	error if too many digits
	BM	CMDERR	error if no digits
	CLHI	R0,256	
	BNL	CMDERR	error if number is too big
	LIS	WORK,X'0D'	check for carriage return
	CLB	WORK,0(PNTR)	
	BNE	CMDERR	
	STB	R0,RWDPBLK+1	
	SVC	1,RWDPBLK	rewind the lu
	....		continue

Before stipulating "skip leading blanks", the number of leading blanks in the buffer must be considered, as the buffer may contain all blanks. The following resolves this problem:

SVC	2,PACKDECS
CLAI	PNTR,BUFFEND
BNL	CMDERR

Another problem is now encountered assume that <lu> were optional, with a default of ZERO. In that case, the BM CMDERR instruction should be omitted since the Pack call sets Register 0 to ZERO if no digits are found.

File Descriptors

This section does not apply to programs which do not perform any file or device handling; however, for systems using LU instead of device, this section points out the advantages of file and device handling. Under OS/16 MT2 it is easier to handle device and file names than it is to handle simple decimal numbers, because the system can do more error checking and default processing.

The Pack File Descriptor call is used to handle device or file names.

When Pack File Descriptor is called, it expedites decoding the entire syntax, including file name, volume name, extension, and separators, handles the optional cases and flags them; and expedites handling one common default case. The format of the parameter block is:

```
PACKFD       DB      0,16,0,PNTR           ordinary call
             DC      A(SV7PB+8)

PACKFDS      DB      X'40',17,0,PNTR       skip leading blanks
             DC      A(SV7PB+8)

PACKFDN      DB      X'80',16,0,PNTR       no volume default
             DC      A(SV7PB+8)

PACKFDNS     DB      X'C0',16,0,PNTR       no default, skip blanks
             DC      A(SV7PB+8)
```

As an example of how this call may be used, the syntax of the rewind command may be changed to:

<rwdcmd> .:= REWIND <fd>

A valid processing routine is:

```
REWIND       SVC     2,PACKFDS            pack fd and skip leading blanks
             BO      CMDERR               branch on syntax error
             CLAI    PNTR,BUFFEND
             BNL     CMDERR               error if scan past end of buffer
             SVC     7,SV7PB              open the device or file on LU 0
             SVC     1,RWD                rewind the device
             SVC     7,CLOS0              close LU 0
             ....                         continue
SV7PB        DB      X'40',0.0,0          open LU 0
             DC      Y'0'
             DS      16                   File Descriptor field
CLOS0        DB      X'04',0,0,0          close LU 0
RWD          DB      X'C0',0,0,0          rewind LU 0
```

In this example, SVC 1 or SVC 7 calls were not checked for errors, since the purpose of this demonstration does not concern use of SVC 1 or 7. A more difficult example is:

<inputcmd> : · =   INPUT <fd>

where the default volume name must be FRED and the default extension must be OBJ Assume the verb INPUT has been detected and that the separator has not yet been checked.

```
WORK         EQU     14
             ....
INPUT        LB      WORK,0(PNTR)         check separator
             CLHI    WORK,C' '
             BNE     CMDERR
             AIS     PNTR,1               separator ok, increment pointer
             LM      WORK,VOL             setup default volume
             STM     WORK,SV7PB+8
             SVC     2,PACKFDN            pack fd , no skip, no default
             BO      CMDERR               branch on syntax error
             BNC     PROCESS              branch unless default extension
             LM      WORK,EXT             setup default extension
             STM     WORK,SV7PB+20
PROCESS      .....                        continue
VOL          DC      C'FRED'
EXT          DC      C'OBJ'
```

The default volume name, FRED, was stored in the receiving area before the SVC 2 call. If the command string contained a volume name, FRED is overlaid. If no volume name was present in the command string, since "no default" was specified, the volume name field of the receiving area is left unchanged. The extension default must be handled following an SVC 2 call

The "no default" option is explained as follows· In any OS/16 MT2 system that has Direct-Access storage, there is a volume called the system default volume. This volume name may be changed by the console operator. Most commands to OS/16 MT2 itself assume that if no volume name is specified, the default volume is assumed. Therefore, the normal call to Pack File Descriptor stipulates default volume. If no volume ID is present in the File Descriptor string, OS/16 MT2 returns the name of the default volume, whatever it may be at the time, as though the user had typed it in.

The reason for the no-default option is now clear There are instances when the program may not wish to default to the name of the default volume, but rather to the last volume name that the user did type in, or perhaps to a volume name that the program specifically set up. If the program specifies no-default, the Pack File Descriptor routine assumes that the program has already set up the volume field of the receiving area for default condition. If the user does not enter a volume ID and the program has specified no-default, the volume field of the receiving area is left unchanged.

The address field of the sample parameter block is defined as A(SV7PB+8). This is because, when the PACK FILE DESCRIPTOR is called, the program desires the receiving area to be the File Descriptor field of an SVC 7 parameter block. This area is located 8 bytes past the start of the SVC 7 parameter block. It is not necessary to move the data directly into a parameter block. Note, however, that the receiving area must be 16 bytes long and must begin on a halfword boundary.

This call sets condition codes. The 'V' flag (CC=4) denotes a syntax error. The 'L' flag (CC=1) denotes no volume ID. This flag is set whether or not no-default is specified. The 'C' flag (CC=8) denotes no extension. If no extension is found, the system automatically gives a default extension of blanks If use of a different default extension is desired, for example, OBJ, the 'C' flag tells when to use it. If the blank extension is satisfactory as a default, the 'C' flag may be ignored. The 'V' flag should never be ignored, since the receiving area is undefined following a syntax error.

Following a syntax error, the command string pointer is returned unchanged.

Further Mnemonics

Operands within a command may take the form of mnemonics, just as the verb did. These mnemonics are processed in the same way as the verb. Abbreviations may be used. The call to Mnemonic Table Scan is performed in the same way. Normally, a separate mnemonic table is used, however, it is possible to use the same mnemonic table that was used to find the verb.

Notice that there is no provision in the call to Mnemonic Table Scan for the skipping of leading blanks. This means that the program has to perform this function, if it is desired. Also note that register PNTR must be pointing at the first character of the mnemonic to be scanned. This means that PNTR must be adjusted to skip the separator that followed the preceding verb or operand.

ASCII Strings

Even with all the facilities described previously, there are times when an operand does not quite fit any of the normal categories (Hexadecimal or Decimal numbers, File Descriptors, or mnemonics). OS/16 MT2 provides a call to meet these contingencies. This is the Move ASCII Characters call. This call should cover any special cases that cannot be handled by one of the normal calls. It has two options One is used when a fixed number of characters must be moved from the command string to a receiving area This number of characters may not exceed 127. The format of the parameter block for this option is:

```
MOVEF           DB              N,18,PNTR,OUTPTR
```

The number of characters to be moved is indicated by N. PNTR is the current buffer pointer, as in all the preceding calls. OUTPTR is a register that holds a pointer to the location of the characters' placement. After the data is moved, OUTPTR is adjusted to point one byte beyond the last character moved.

The second option is considerably more complex. This option is used when a variable number of characters, terminated by some separator, is to be moved. In this case, the parameter block is·

```
MOVEV           DB              X'80'+N,18,PNTR,OUTPTR
                DAC             SEPSTR
```

The maximum number of characters to be moved is indicated by N. PNTR and OUTPTR are the current buffer pointer registers. SEPSTR is the address of a string of separators. When any of the separators in this string is found in the command string, the move operation is terminated. If none of the separators is found, the move operation terminates when the maximum number of characters (N) has been moved. The Condition Code V flag is set (CC=4) to show that no separator has been found The separator string is coded as follows:

SEPSTR	DB	NUM,SEP1,SEP2,SEP3,...

where NUM is the number of separators in the string and SEP1, SEP2, etc., are the actual separators. For example, if the program wishes the move operation to terminate, if the character * or the character $ were found, the separator string appears as follows

SEPSTR	DB	2,C'*$'

A more possible example depicts the characters (blank), CR (carriage return), , (comma), or ; (semicolon) as valid separators. In this case, the separator string appears as:

SEPSTR	DB	4,C' ,;',X'0D'

The calling method is as follows

LDAI	OUTPTR,OUTBUFF	output buffer address to register OUTPTR
SVC	2,MOVEV	move variable number of characters
BO	NOSEPS	branch if no separator
.. ..		continue

CHAPTER 5
GUIDE TO TASK-HANDLED TRAPS

INTRODUCTION

A complete OS/16 MT2 system encompasses a 16-Bit Processor, the OS/16 MT2 operating system, and a set of tasks This section explores the makeup of a task at the task block structure level A thorough discussion of the writing of the code contained in the executable task structures is beyond the scope of this section For such information the reader is referred to:

> *16-Bit Series Reference Manual*
> Chapters 2 and 3 of this manual
> *Common Assembly Language Users Manual*

The FORTRAN user should refer to the *FORTRAN V Level II User's Manual* for information on representing the various task structures described here in the FORTRAN language.

This section also describes in detail the construction of task data structures using system STRUCS supplied with OS/16 MT2. Examples are given for structures referred to in this section as well as general uses of system STRUCS.

The various steps required to prepare a task for execution are also outlined

TASK STRUCTURE

A task in OS/16 MT2 contains code to do the work of the task (Mainline Code), a set of User Dedicated Locations (UDL), and an initial Task Status Word (TSW). Any program, including a standard utility, that is to be run as a task under MT2, can be ignorant of its UDL and TSW since TET/16, which establishes it as a task, leaves the appropriate amount of space in its image output for the UDL and supplies an appropriate default for the initial TSW.

In addition to the UDL, TSW, and mainline code that a simple task might contain, the various features provided in MT2 require additional task components It should be understood that each additional component is only required to be included in the task if the MT2 facility it supports is to be used. The potential components of a task are as follows

> Task Status Word (other than default)
> User Dedicated Locations
> Mainline Code
> Task Queue
> Task Queue Trap Service Routine
> Power Restoration Trap Service Routine

The following discussions of each component indicate which OS facilities are involved and, therefore. which task components are to be included when writing the task during the task's design process Figure 5-3, at the end of this section, summarizes which task components are required for various OS facilities

Task Status Word (TSW)

The Task Status Word (TSW) describes the state of a task at any one time with respect to user controlled interaction with the operating system The various task traps, as well as the different reasons for additions to the task's Task Queue are controlled through the task's TSW. This status word is used to enable or disable the various task traps, enable or disable additions to the Task Queue from various sources and contains the location counter and condition code upon taking and returning from task traps. TSW bit assignments are shown in Figure 5-1

```
      BIT 0                                              15
BYTE
   0  |            STATUS                                  |
   2  |            LOC                                     |
```
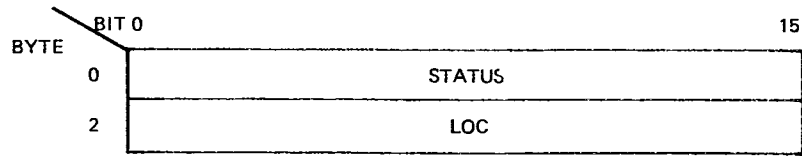
TASK STATUS WORO IS 32-BITS THE FIRST HAl FWORO (BITS 0-15) CONTAINS
STATUS OEFINEO IN FIGURE 5-1. SECONO HAL:-WORO (BITS 16-31) CONTAINS
THE CURRENT LOCATION COUNTER AS IN A P' W.

STATUS BITS			
BIT	HEX MASK	NAME	OESCRIPTION
0	8000	W	TRAP WAIT — TASK SUSPENOEO UNTIL A TASK TRAP OCCURS
1	4000	P	POWER RESTORATION TRAP ENABLE — A TASK TRAP OCCURS ON COMPLETION OF SYSTEM POWER RESTORE SEQUENCE
2-3			RESERVEO — MUST BE ZERO.
4	0800	Q	TASK QUEUE SERVICE TRAP ENABLE — A TASK TRAP OCCURS WHEN AN ITEM IS AOOEO TO THE TASK QUEUE. ALSO A TRAP OCCURS IF A TSW IS LOAOEO WITH THIS BIT SET ANO THE TASK QUEUE IS NOT EMPTY.
5			RESERVEO — MUST BE ZERO
6	0200	O	ENABLE TASK QUEUE ENTRY ON INTERRUPT FROM A CONNECTEO TRAP GENERATING OEVICE.
7	0100	T	ENABLE TASK QUEUE ENTRY ON QUEUE PARAMETER (SVC 6) REQUEST OIRECTEO AT THIS TASK.
8	0080	Z	ENABLE TASK QUEUE ENTRY ON COMPLETION OF A TIME INTERVAL.
9	0040	O	ENABLE TASK QUEUE ENTRY ON COMPLETION OF I/O PROCEEO REQUEST.
10			RESERVEO
11	0010	E	ENABLE TASK QUEUE ENTRY ON SENO MESSAGE
12-15	000F	CC	CONOITION COOE (AS IN PSW).

Figure 5-1. Task Status Word (TSW)

The task has a current TSW at all times. The initial TSW of the task (the current TSW upon loading the task) is set at Task Establishment time At that time a specific Initial TSW may be set; otherwise, the intial TSW defaults to zero (no traps or queue entries enabled), plus a location counter of the starting address specified at assembly time (argument of the END assembly statement) or, if unspecified, the start of the task code.

When a task trap occurs (that is, when a trap causing condition occurs while the particular trap is enabled in the current TSW) the current TSW is saved in the appropriate area of the User Dedicated Locations (UDL) and a new TSW is loaded from the appropriate area of the UDL. The new TSW controls the traps or task queue entries that are to be allowed during the execution of the trap service routine.

A task may change its current TSW at any time by executing an SVC 9 load TSW. The argument address of the SVC 9 call points to a four byte area aligned on a halfword boundary, containing the new TSW to be loaded.

The first two bytes contain the status and the next two contain the location counter of the TSW Following SVC 9 processing, the task resumes execution at the location specified in the loaded TSW. If only the status of the current TSW is to be changed, without affecting the flow of execution of the task, a zero location counter should be specified in the new TSW. In this case, execution resumes at the instruction following the SVC 9 call.

If an SVC 9 call loads a TSW enabling a task trap that is capable of occurring immediately (e.g , enabling Task Queue Service Trap while the Task Queue is not empty), a TSW swap occurs, storing the just loaded TSW in the UDL and loading a new TSW from the UDL.

Also, an SVC 9 load of a TSW enabling Trap Wait places the task in Trap Wait, suspending execution until one of the traps enabled in the same TSW occurs. Since execution never resumes with the Trap Wait bit set, the location counter portion of such a TSW is ignored.

NOTE

If a task is in Trap Wait with no task traps enabled, it will wait indefinitely or until it is cancelled

User Dedicated Locations (UDL)

A task's UDL starts at the first location within its partition. It contains Task Status Word (TSW) swap areas and other data used for communication between the operating system and the task, as follows (refer to Figure 5-2). Items in parentheses are the names of the relevant fields in the UDL STRUC, to be discussed later:

CTOP (UDL CTOP)	Contains the program address of the highest halfword in the task's allocated memory.
COMBOT (UDL.CBOT)	Contains same address as CTOP. Provided for compatibility with older 16-Bit Operating Systems.
UTOP (UDL.UTOP)	Contains the program address of the first halfword following the user's program
UBOT (UDL.UBOT)	Contains the program address of the bottom of the user's program.
Power Restore Old TSW (UDL.PWRO)	Set by the operating system to the task's current TSW on occurrence of a Power Restoration Trap.
Power Restore New TSW (UDL.PWRN)	Set by the user to the TSW to be loaded on the occurrence of a Power Restoration Trap. The LOC portion of this TSW points to a Power Restoration service routine.
Task Queue Service Old TSW (UDL TSKO)	Set by the operating system to the task's current TSW on occurrence of a Task Queue Service Trap. A Task Queue Service trap occurs when task queue service trap is enabled in the current TSW and the Task Queue is non-empty.
Task Queue Service New TSW (UDL.TSKN)	Set by the user to the TSW to be loaded on the occurrence of a Task Queue service trap. The LOC portion of this TSW contains the address of a Task Queue service routine.
A (Task Queue) (UDL.TSKQ)	Set by the user to the address of the task's Task Queue This location must be set prior to enabling any entries to the Task Queue If the contents of this location is zero, no task queue entries are made regardless of the state of the Task Status Word's queue entry enabling bits (TSW.DIQM, TSW TCM, TSW.IOM, TSW.TMCM).
A(Message Buffer) (UDL.MSGR)	Set by the user to the address of the first (or only) message buffer, if messages are to be received from another task in the system If this item is zero, no messages can be received.
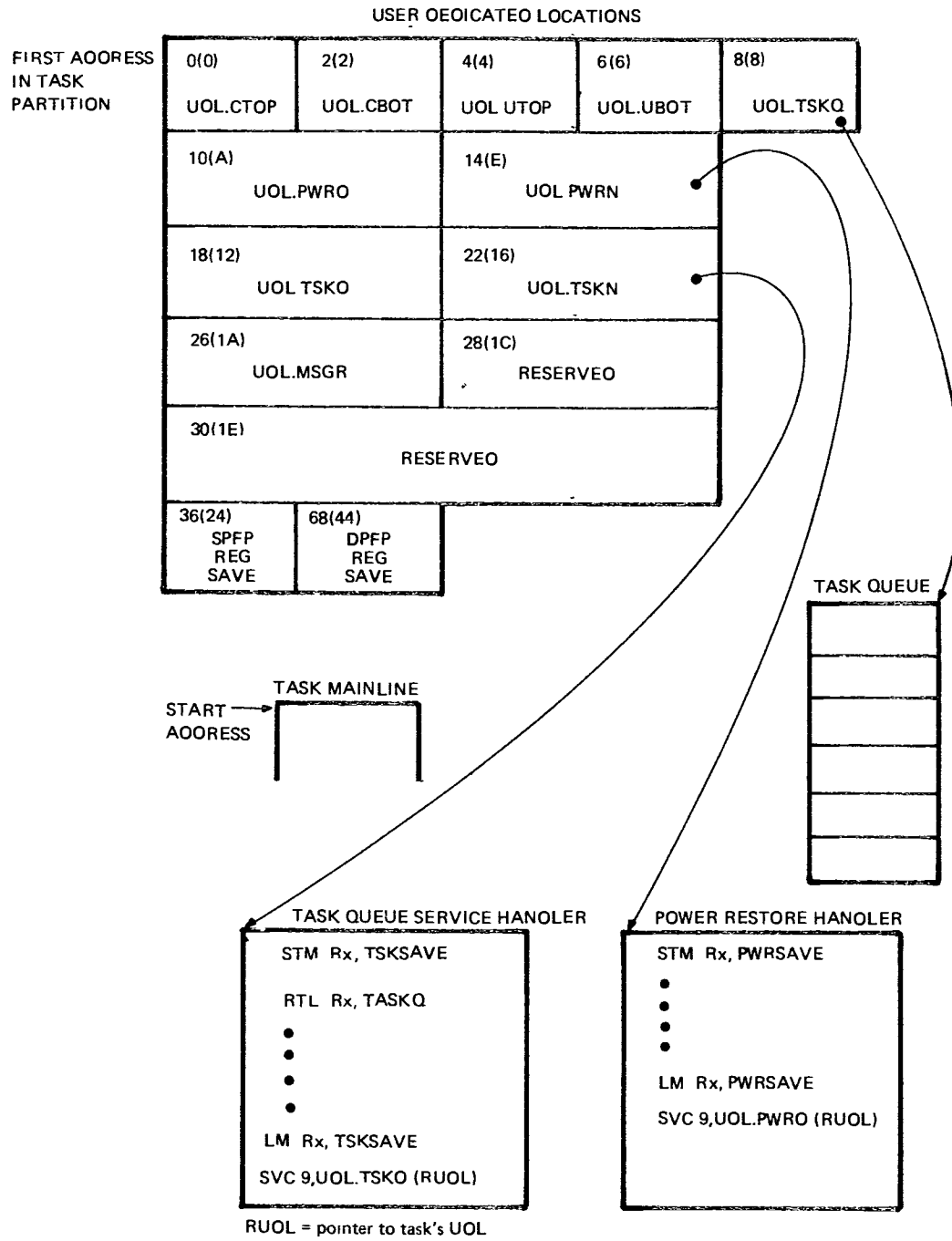
Figure 5-2. Task Structure

For the user supplied fields in the UDL (UDL.TSKQ, UDL.PWRN, UDL.TSKN, UDL.MSGR) there is a choice between assembling the UDL containing these fields as constants or loading these fields during the task initialization phase. After a program is assembled into object code, it must be established as a task with TET/16

If a task includes an assembled UDL, the command OPTION NOUDL must be passed to TET/16 at task establishment, so that TET/16 does not reserve a UDL area on behalf of the task

If a task sets up its UDL area dynamically at run-time, no special OPTION command is necessary. TET/16 automatically reserves and initializes the UDL to zero; 132 bytes if OPTION DFLOAT is specified, 68 bytes if OPTION FLOAT is specified, or 36 bytes if no floating point support is required.

Mainline Code

If the task is an autonomous program with no need of the task trap features of OS/16 MT2, the mainline code consists of the original program that is being run as a task. Together with its UDL, it is a complete task

However, when the task makes use of the task trap features of the OS, the exact function of the mainline code is a basic design choice. Mainline task code can be nothing more than a short start-up routine with all the work of the task being done in trap service routines or it can perform the bulk of the work with only small amounts of work done in the service routines. The particular design depends on the task's application.

Task Queue

The operating system uses the Task Queue to inform a task of the occurrence of the events shown in Table 5-1. A Queue is used so that entries (referred to as items) resulting from events are not lost during the time the task is servicing a previous item. A Task Queue is formed by the assembly language statements:

```
label    DB n,0,0,0
         DS n*2
```

Operand 'n' is the size (number of items capacity) of the queue, and label is used as the address of the Task Queue. UDL.TSKQ, in the UDL, must be set to this address before entries can be made to the Task Queue (see Figure 5-2). No specific advice is given for the size of the queue since this decision depends on the task's application and expected frequency of occurrence of queue-related events compared to the individual event service time. In general, if a queue related event occurs when the queue is full, the associated item is lost. Therefore, the following guidelines should be used in determining a *minimum* queue size:

    1 entry for each device capable of interrupting concurrently
    n entries for each task capable of queueing n parameters concurrently
    1 entry for each Logical Unit having concurrent I/O proceeds
    1 entry if a time interval trap is to be scheduled (SVC 2 code 23)
    m entries for each task capable of sending m messages concurrently

A Task Queue Service Trap occurs whenever the Task Queue Service Trap enable bit (Bit 4, symbol TSW.TSKM) is set in the current TSW *and* the Task Queue is non-empty (i.e., has at least one item in it). Items are added to the queue only if the particular queue entry enabling bit is set in the current TSW (shown in Table 5-1).

The Task Queue is a standard INTERDATA 16-Bit Series circular list (refer to the *16-Bit Processor User's Manual,* Chapter 3, for a detailed description). The operating system adds items to the bottom of the queue, therefore, the task would typically remove items for servicing from the top of the queue. For example:

    RTL R1,TASKQ

(an item is removed from the top of the Task Queue at TASKQ and placed in register R1 for examination/processing).

The user can use the ABL machine instruction to add simulated items to the queue before enabling Task Queue Service Traps, thereby simulating a Task Queue Service Trap, perhaps to start off a cycle of I/O Proceeds.

Resident Tasks that are to be cancelled and restarted by other tasks should initialize their Task Queues to an empty state each time they are restarted to prevent old queue items from causing erroneous behavior by the task. An example of code to initialize a Task Queue at address TASKQ is:

    LIS   RF,0
    STB   RF,TASKQ+1
    STH   RF,TASKQ+2

TABLE 5-1. TASK QUEUE ITEMS

SOURCE	QUEUE ENTRY ENABLING BIT*	CONTENTS OF ITEM
Oevice Interrupt	X'0200' (TSW.OIOM) *	Parameter associated with device
SVC 6 Queue Parameter	X'0100' (TSW.TCM) *	Parameter specified in call
Timer Termination	X'0080' (TSW TMCM) *	Parameter specified in call
I/O Proceed Completion	X'0040' (TSW IOM) *	Address of SVC 1 Parameter Block
Task message Received	X'0010' (TSW PMM) *	Address of message buffer

* The name in parentheses is the symbolic name of the mask used to enable the associated bit (to be discussed later).

Task Trap Service Routines

A Task Trap occurs when a particular Task Trap is enabled in the current TSW at the time of a trap causing event Table 5-2 gives the types of Task Traps, enabling bit in the current TSW, and Trap Causing event.

TABLE 5-2. TASK TRAPS

TASK TRAP	ENABLING BIT*	EVENT
Power Restoration	X'4000' (TSW.PWRM) *	System Power Restoration
Task Queue Service	X'0800' (TSW.TSKM) *	Task Queue Non-Empty

* The name in parentheses is the symbolic name of the mask used to enable the associated bit.

A trap service routine must be written and included in the task for each task trap to be used Prior to enabling the Task Trap, the TSW swap area in the UDL for that trap must contain a new TSW for that trap (see Figure 5-2). The status portion of the new TSW is the Task Status to be in effect during the execution of the trap service routine; the location counter portion is the entry address of the trap service routine. Care must be exercised in enabling any Task Traps in the new TSW status of a particular Task Trap. (If two Trap Service routines, A and B, enabled each other's Trap and Trap A occurred, Trap B might occur enabling Trap A again and if Trap A occurs. the first old TSW for Trap A would be lost.)

Since a Task Trap may occur during execution of code elsewhere in the task it is the responsibility of each Task Trap service routine to save any registers it uses to avoid disturbing the interrupted code.

Task Queue Service Routine

This user written routine is entered upon a Task Queue Service Trap (i.e., whenever Task Queue service is enabled in the current TSW and the Task Queue is non-empty) The Task Queue Service new TSW field of the UDL (that is. the TSW in effect during Queue Service) typically has a status disabling Task Queue Service (if enabled, the Task Queue Service routine could interrupt and reenter itself with dubious consequences) but enabling various queue entries to continue while the trap-causing item is serviced.

Table 5-1 shows the five types of events that can cause items to be added to the task queue. Each item is 2 bytes in length with the content dependent on the particular item as follows:

Device Interrupt The item is the parameter specified at the time the Trap-Generating Device was CONNECTed (via SVC 6) to the Task.

SVC 6 Queue Parameter: The item is the parameter specified in SVC 6 parameter block causing a parameter to be queued to this task.

I/O Proceed Completion The address of the SVC 1 parameter block of a completed I/O proceed.

Interval Completion Termination: The item is the parameter specified in the SVC 2 code 23 which caused the Timer Termination.

Task Message Received: The item is the address of the message buffer

The item is removed from the Task Queue with an RTL instruction After the trap-causing item is decoded and the appropriate action taken, the Task Queue Service routine typically exits with an:

<center>SVC 9,UDL.TSKO(RUDL)</center>

where UDL.TSKO is the symbolic name of the UDL Task Queue Service old TSW save area (to be discussed later) and RUDL is a pointer to the task's UDL. By loading the old TSW in this way, execution resumes at the location, with the status in effect, at the time the task was trapped. If additional items are on the Task Queue, loading the old TSW causes the Task Queue Service trap to recur for service of the additional items.

Power Restoration Trap Service Routine

This user written routine is entered upon a Power Restoration Trap (whenever Power Restoration trap is enabled in the current TSW and a system power restoration takes place, note that if the Power Restoration Trap is disabled at the time of a system power restoration, the task is PAUSEd. Once this routine is entered, it can go about the business of recovering the task from the power outage (highly application dependent), and if it determines the situation warrants resumption of the task, the routine exits with:

<center>SVC 9,UDL.PWRO(RUDL)</center>

Figure 5-3 summarizes task requirements.

REQUIRES OS FACILITY	NON-ZERO TSW	UDL	TASK QUEUE	TASK QUEUE SERVICE RTN.	PWR RESTORE SERVICE RTN.	MESSAGE BUFFER
SVC 2 CODE 5 FETCH POINTER		X				
RECEIVING SVC 6 QUEUE PARAMETERS	X	X	X	X		
RECEIVING PARAMETER ON OEVICE INTERRUPT	X	X	X	X		
RECEIVING PARAMETER ON TIMER COMPLETION	X	X	X	X		
RECEIVING PARAMETER ON I/O PROCEEO COMPLETION	X	X	X	X		
RECEIVING MESSAGE	X	X	X	X		X
POWER RESTORATION TRAP	X	X			X	

<center>Figure 5-3. Task Requirements Summary</center>

OS/16 MT2 SYMBOLS AND STRUCS

The CAL Assembler provides a powerful facility for referencing, by name, numerical constants and constant displacements within a data structure (STRUCs) (refer to the CAL User's Manual, for detailed syntactical description of STRUCs). A reference made in such a way is said to refer symbolically to a constant (or constant displacement).

Symbolic references are generally much easier to use than their numeric counterparts because of the ease in remembering a name Errors involving references are less likely since using an inaccurate numerical constant is usually assembled without an error while using an inaccurate symbolic name is caught by the assembler (as an undefined symbol). Symbolic references help make assembly language code easier to read and understand because symbolic names can be chosen to reflect the meaning of the numerical constant represented or the name of the field within a data structure which the symbol points to Lastly, in the event that a numerical constant or displacement is ever changed, code using symbolic references is easier to update since the update necessitates changing only the symbol definition rather than every reference (as would be needed with numerical references).

Numerical constant symbols and displacement symbols (STRUCs) are used throughout the coding of OS/16 MT2 and these same symbols and STRUCs are equally usable by assembly language programmers writing programs to run under MT2. The collection of symbols and STRUCs related to OS/16 MT2 is contained in the Parameters and Control Blocks module supplied with the system source (filename PCB16.CAL for MT on disc). Some examples of uses by user tasks follow

1. Construction of a Task Status Word:

   The status portion of a Task Status Word (TSW) enabling Trap wait, Task Queue service trap, Queue entry on Task call (queue parameter from another task) and I/O proceed termination can be written

   ```
   COPY          UDL
                 .
                 .
                 .
   DC            TSW.WTM!TSW.TSKM!TSW.TCM!TSW.IOM
   ```

   This expression instructs the assembler to OR together masks, each setting a particular bit, to form a word with all the required bits set. (Within the Parameters and Control Blocks module symbol definitions, symbols ending in 'M' have the value of the bit *mask* needed to enable a particular bit.)

2. Loading a TSW into the User Dedicated Locations (UDL)·

   A TSW enabling queue entry on I/O proceed termination and timeout completion is to be loaded into the "Task Queue service new TSW" field of the User Dedicated Locations, thereby allowing queue entries for these two events to continue while the Task Queue is being serviced. The TSW in question contains a LOC field pointing to the Task Queue service routine.

   ```
               COPY      UDL
               .
               .
               SVC       2,FETCH                  GET ADDRESS OF UDL INTO R13
               LM        R14,TSKQTSW              LOAD TSW INTO REGS 14-15
               STM       R14,UDL.TSKN(R13)        STORE INTO UDL
               .
               .
   QSERVICE    EQU       *                        TASK QUEUE SERVICE ROUTINE
               .


   FETCH       DB        0,2
               DC        13
   TSKQTSW     DAC       TSW.IOM!TSW.TMCM,QSERVICE
   ```

   Note that the one line definition of the TSW (TSKQTSW) generates two halfwords· a STATUS portion enabling certain bits and a LOC portion pointing to the task queue service routine as QSERVICE

3   Loading an SVC 6 function code into an SVC 6 parameter block·

A function code specifying Load and Start immediately for some "other" task, (as opposed to a self-directed SVC 6) is loaded into an SVC 6 parameter block.

```
              COPY      SVC6.
              .
              .
              .
              LHI       R14,SFC1.DOM!SFC1.LM        LOAD FUNCTION
              LHI       R15,SFC2.SIM                START FUNCTION
              STM       R14,PARBLK+SVC6 FUN         STORE INTO SVC 6 P.B.
              .
              .
              .
PARBLK        EQU       *                           SVC 6 PARAMETER BLOCK
              DS        SVC6.
```

Note that the proper amount of space for the SVC 6 parameter block was reserved by a "DS SVC6." where "SVC6." is the label of the STRUC defining the SVC 6 parameter block and set by CAL to the size of the data structure so defined.

A field within a data structure can be referenced in one of two ways using a STRUC:

```
              COPY      SVC6.
              .
              .
              .
              STM       R14,PARBLK+SVC6.FUN
              .
              .
              .
              DS        SVC6.
```

or referencing a field through an index register·

```
              COPY      SVC6.
              .
              .  .
              .
              LDAI      R3,PARBLK
              .
              .
              .
              STM       R14,SVC6.FUN(R3)
              .
              .
              .
PARBLK        DS        SVC6.
```

The direct method has the advantage of not using an additional register while the index register method has the additional flexibility of passing the address of different parameter blocks through a register (perhaps to a subroutine).

The previous examples deal with applications where a program dynamically loaded various fields with appropriate values. This approach is correct for occasions where the contents of the various fields change with time and therefore, must be dynamically initialized and subsequently changed. However, for applications where the contents of fields remain the same for the duration of the task, it saves both assembly code size and execution time to assemble the appropriate values into the data structures. Some examples follow.

4   Assembling values into a UDL (assembling a UDL as part of a task requires an OPTION NOUDL statement in the TET/16 input at task establishment time):

```
           COPY    UDL
TUDL       EQU     *
           ORG     TUDL+UDL.TSKQ
           DC      task queue addr          ADDRESS OF TASK QUEUE
           ORG     TUDL+UDL.PWRN
           DC      status,loc               POWER RESTORE NEW TSW
           ORG     TUDL+UDL.TSKN
           DC      status,loc               TASK QUEUE SERV NEW TSW
           ORG     TUDL+UDL
```

Note that the label 'TUDL' is used so as not to conflict with 'UDL' which is defined in the UDL STRUC (invoked by the COPY UDL). To omit any of the above field definitions from the code simply delete the ORG to the relevent field plus the constant definition for that field. The 'ORG TUDL+UDL' sets the location counter past the end of the UDL.

5   Assembling Task ID and function code into an SVC 6 parameter block at PARBLK:

```
           COPY    SVC6.
           .
           .
           .
PARBLK     EQU     *
           ORG     PARBLK+SVC6.ID
           DC      C'TASKA '                TARGET TASKID
           ORG     PARBLK+SVC6.FUN
           DC      SFC1.DOM,SFC2.SIM        START IMMED.FUNCTION
           ORG     PARBLK+SVC6.
```

NOTE

The ORG PARBLK+SVC6. sets the location counter past the
end of the Parameter block.

STRUCs are provided for SVC 1, SVC 5, SVC 6, and SVC 7 parameter blocks, and the User Dedicated Locations. They can be used after a CAL 'COPY' of the relevant name (a COPY statement instructs CAL to include source from Logical Unit 7, in this case the Parameters and Control Blocks module):

STRUC	COPY
SVC 1	SVC1
SVC 5	SVC5.
SVC 6	SVC6.
SVC 7	SVC7.
UDL and TSW Bits	UDL

Refer to Appendix 3 for assembly listings of the complete Parameters and Control Block module.

NOTE

Certain fields and bits in the UDL, TSW and SVC 6 STRUCs are
included for future extensions of the Operating System and should
be ignored by users of R00 and R01 OS/16 MT2.

TASK PREPARATION

A task must go through various transitions between the time it is written by the programmer in assembly language (or a combination of FORTRAN and assembly language or FORTRAN alone) and the time it is run under OS/16 MT2. These steps include the following (refer to Figures 5-4, 5-5, and 5-6):

FORTRAN Compiler - if required, this compiles the FORTRAN source plus any imbedded assembly language into CAL source (see the *FORTRAN V Level 1 Reference Manual, FORTRAN V Level 1 Users Guide,* and *FORTRAN V Level 1 Run Time Library Manual* for details).

CAL Assembler - This translates the CAL assembly language into 16-Bit object format output (see the *Common Assembler Language Users Manual*).

TET/16 - This converts the object format output of CAL into memory-image format References by the TASK to a resident library partition are resolved with TET RESOLVE command (Figures 5-4 and 5-6). Relevant routines from the library can be included in the task itself with a TET EDIT command (Figure 5-5). (See the *OS/16 MT Operators Manual,* Chapter 5, for a complete discussion of TET with additional examples.)

OS/16 MT2 - The OS/16 MT2 loader loads the task or library (via operator command or SVC 6) that was established by TET, and allows it to run on a 16-Bit Processor.

NOTE

For a task to use TASK COMMON, it must contain a labelled Common with the Label 'TSKCOM' at assembly time (the left-hand CAL box in Figure 5-4). All program references to this common block are resolved by TET/16 to a reference into user specified TASK common partition.

Figure 5-4. CAL Source

FORTRAN
SOURCE

(FORTRAN SOURCE)

FORTRAN
COMPILER

FORTRAN
RTL

(CAL SOURCE)

CAL

(OBJECT, BROUGHT
IN WITH TET EDIT
COMMAND)

(OBJECT)

TET/16
(ESTABLISH
TASK)

(TASK—IMAGE)
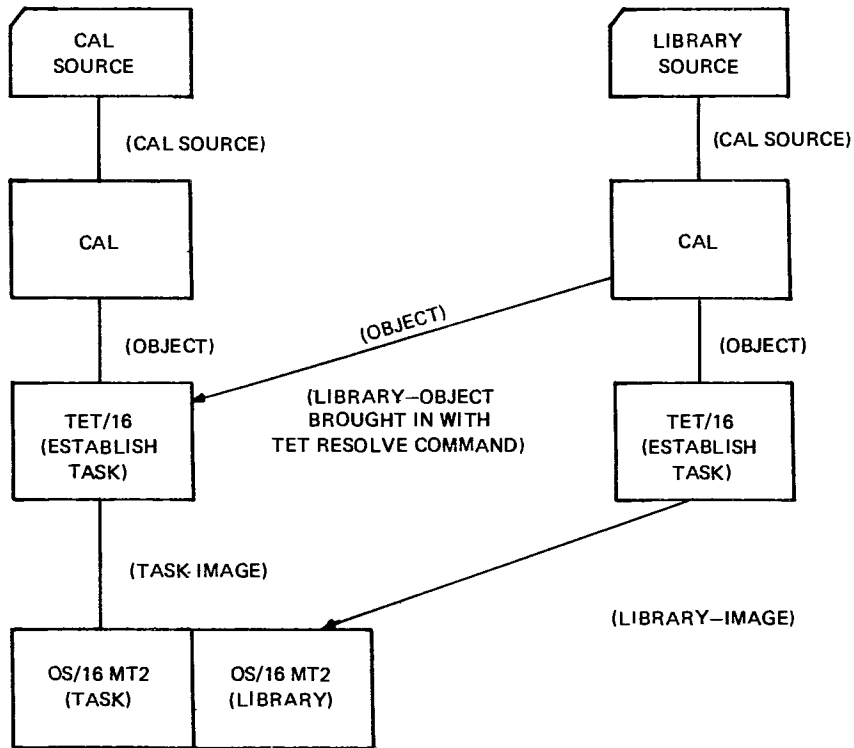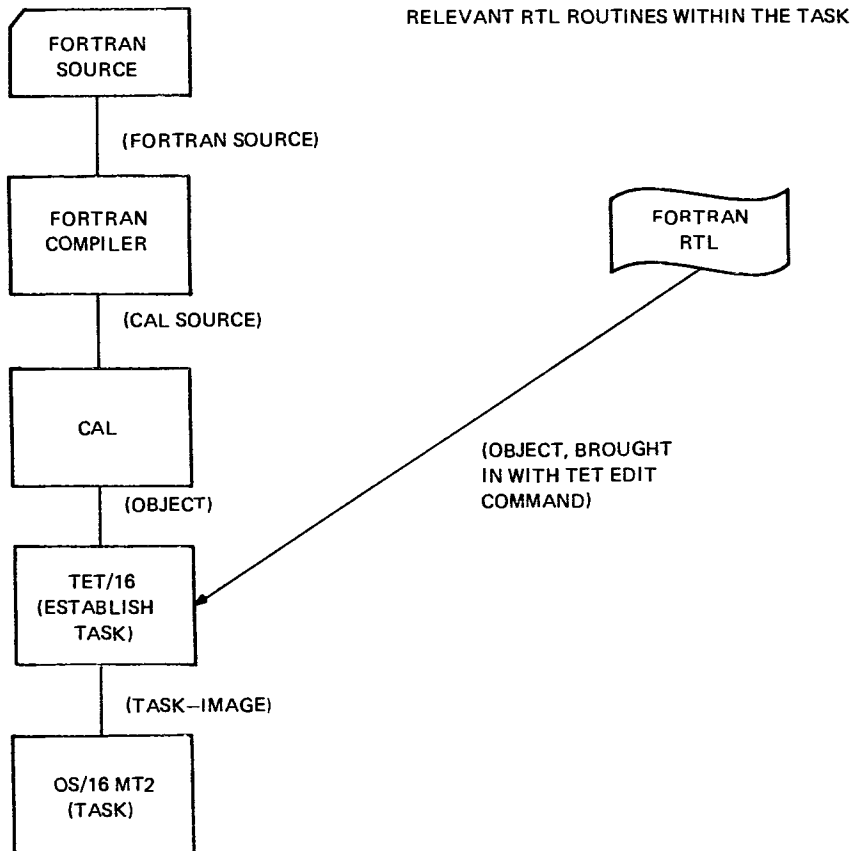
OS/16 MT2
(TASK)

Figure 5-5. FORTRAN Source

RTL IN RESIDENT LIBRARY PARTITION OF OS/16 MT2

FORTRAN
SOURCE

(FORTRAN SOURCE)

FORTRAN
RTL

FORTRAN
COMPILER

(CAL SOURCE)

TET/16
(ESTABLISH RL)

CAL

(RTL—OBJECT
BROUGHT IN
WITH TET
RESOLVE
COMMAND)

(OBJECT)

TET/16
(ESTABLISH
TASK)

(TASK—IMAGE)

(LIBRARY—IMAGE)

OS/16 MT2
(TASK)
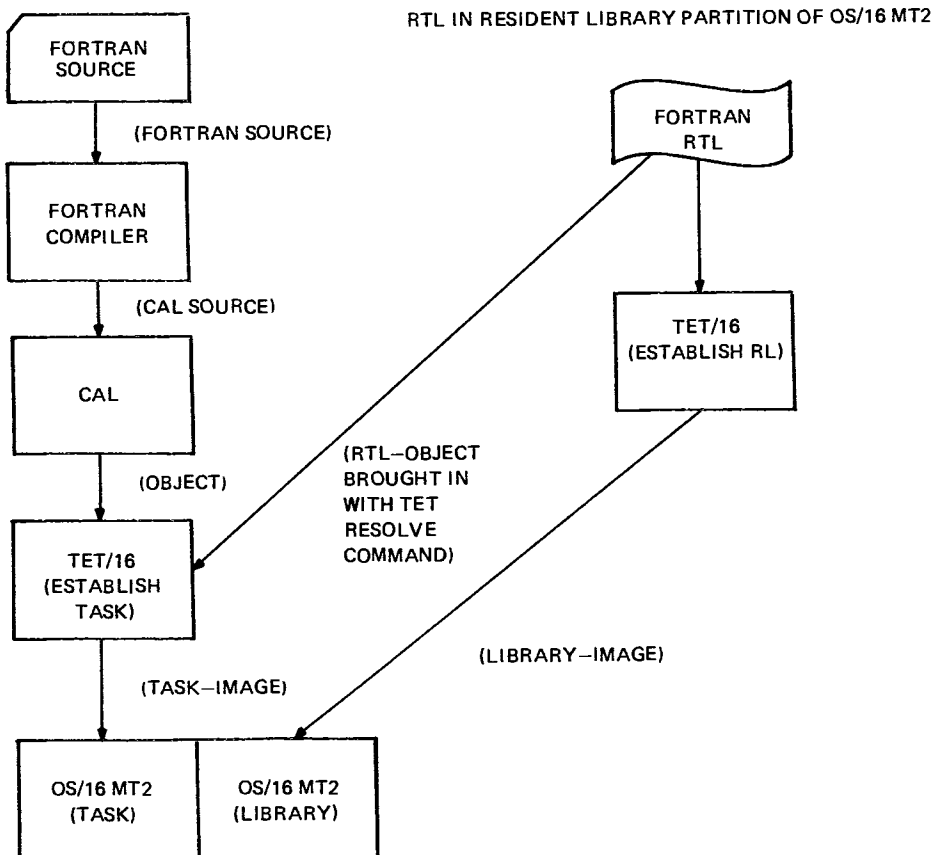
OS/16 MT2
(LIBRARY)

Figure 5-6. FORTRAN Source With RTL

CHAPTER 6

GUIDE TO OS/16 MT2 FILE STRUCTURES

DEFINITION OF TERMS

Certain terms, which are used in this section, must be defined in order to gain a good understanding of the concepts to be presented later These definitions should be read by even the advanced reader, since the same terms have been used throughout the data-processing industry with unfortunate, inconsistent meanings. Refer to Chapter 6 of the *System Planning and Configuration Guide* for definitions.

FILE IDENTIFICATION

The OS/16 MT2 file is identified by a File Descriptor. The full name of a file has three parts: volume name, filename, and extension. The volume name is composed of one to four alphanumeric characters, of which the first character must be alphabetic. This is the name of the volume on which the file resides.

The filename consists of one to eight alphanumeric characters, of which the first must be alphabetic. This is the main identifier for the file, and may be anything the user chooses.

The extension consists of up to three alphanumeric characters. It may consist of no characters at all, in which case it is considered to be made of blanks. The extension usually denotes the type of material on the file. It may be anything the user chooses; however, some specific extensions are used by OS/16 and by some OS utilities, and are assumed to have specific meanings. These extensions are:

TSK	Task Image format
OBJ	Absolute or Relocatable loader format
FTN	FORTRAN source format
CAL	CAL assembly language source format
BAS	BASIC source format
ROL	Roll file for Roll In/Out
CSS	Command Substitution System source format
SPL	Spooled printer listing format
PRT	Spooled printer listing format
MAC	CAL Macro source format

Other extensions may be defined from time to time as required. The user may use any of these standard extensions, or may define others.

File Descriptors are written as follows:

VOLN:FILENAME.EXT

where VOLN is the volume name FILENAME is the filename, and EXT is the extension. VOLN is often omitted in writing and in OS commands. EXT is sometimes omitted, in cases where the extension is already known.

Some examples of valid File Descriptors are ·

FRED:PROGRAM.CAL	A program named PROGRAM, in CAL source format, on volume FRED.
FRED.PROGRAM.OBJ	The same program, in loader format.
FRED:PROGRAM.TSK	The same program in task image format.
GOLF BOGEY.XYZ	The volume now is GOLF; the extension XYZ is not standard, so no assumptions can be made about format
FLAH BOGEY.XYZ	The same file on a different volume. Although all filename/extension combinations on any one volume must be unique, the same filename/extension may exist on more than one volume.

LULU.	The volume name is omitted and should be identifiable in the proper context. The extension field is blank.
VOL3.DATABASE	The extension is omitted here. This may not be the same as the blank extension, but the true extension should be identifiable from context.

Note that when the volume is specified, a colon (:) is always present. When the extension is specified, a period (.) is always present.

Some invalid File Descriptors are

2V.ABC.CAL	Volume name begins with a numeric
V3·FILE #2 OBJ	File name contains a non-alphanumeric
SYSVOL:FILE.DMP	Volume name too long
SV·.EXT	File name nonexistent
PROGRAM325	File name too long

A file is originally named when it is allocated. At this time the specified name is checked to see if it is unique on the given volume. A file can be renamed at a later time, either by the console operator or by a user program. At this time, the new name is checked to make sure it, too, is unique

The renaming function can be used, for example, to handle the updating of a file, when it is desired to retain the older version of the file. One way to do this might be

(Assume the file to be updated is named FILE.XYZ)

1. Rename FILE.XYZ to be FILE BAK
2. Allocate a new file named FILE.XYZ
3. Copy FILE.BAK onto file FILE.XYZ, adding update information as required.

Now there is a most-recent version named FILE.XYZ, and a next-most-recent version named FILE.BAK. When the backup is no longer needed, FILE.BAK may be deleted from the volume

FILE PROTECTION

Files may be protected in one of two ways: statically or dynamically.

Static Protection

Each file has two protection keys, one for Read access and one for Write access. These keys are first set up when the file is allocated. Each key is one byte long and may have any value.

If the values of the keys are within the range $X'01'$ to $X'FE'$, the file may not be assigned for Read or Write access unless the assigning task matches the appropriate keys. In other words, if a file's Read key is $X'3A'$ and its Write key is $X'57'$, a task trying to assign the file for Read access must supply a Read key of $X'3A'$. To assign this file for Write access, a Write key of $X'57'$ must be supplied. In order to assign the file for both Read and Write access, the task must supply both keys.

The values $X'00'$ and $X'FF'$ have special meanings. If a key has a value of $X'00'$, the file is unprotected for that access mode. Any key supplied by the assigning task is accepted as valid. If both keys have a value of $X'00'$, therefore, any attempt to assign the file is accepted, regardless of the keys supplied by the assigning task.

If a key has a value of $X'FF'$, the file is unconditionally protected for that access mode. It may not be assigned for that access mode by any task, regardless of the key supplied by the assignment task. However, an E-task (an executive task, with the same privileges as the operating system itself) may assign any file, even if its protection keys are set to $X'FF'$. Static protection does not apply to E-tasks

Some examples of Static protection are:

Write Key	Read Key		Meaning
00	00	....	Completely unprotected.
FF	FF	....	Unconditionally protected.
07	00	...	Unprotected for Read, conditionally protected for Write (user must supply Write key = $X'07'$).
FF	A7	...	Unconditionally protected for Write, conditionally protected for Read.
00	FF	....	Unprotected for Write, unconditionally protected for Read.
27	32	....	Conditionally protected for both Read and Write.

Dynamic Protection

While a task is using a file, it may wish to prevent other tasks from accessing that file. An example might be a task that is updating a shared data base. In most cases, it would not be appropriate to let the data base be read while the updating operation is under way. For this reason, the task may ask for exclusive access privileges, either for Read or Write, when it assigns a file If it is granted exclusive access, it may be sure that the file is not accessible by any other task until it is closed, or until the task voluntarily gives up exclusive access. This form of protection is called *dynamic* because it is only in effect while the file is actually assigned.

The access privileges are generally known by their abbreviations:

SRO	Shared Read-only
ERO	Exclusive Read-only
SWO	Shared Write-only
EWO	Exclusive Write-only
SRW	Shared Read-Write
SREW	Shared Read, Exclusive Write
ERSW	Exclusive Read, Shared Write
ERW	Exclusive Read-Write

Protection Modification

The protection keys of a file may be changed by the console operator, or by any task having that file assigned for Exclusive Read-Write (ERW).

A task may change its access privileges on a file without having to close the file. This is only possible if the proper conditions are met. A task having a file assigned for shared Read, for example, may not change to exclusive Read if the file is also assigned for shared Read to another task (or another Logical Unit of the same task). Access may always be changed from exclusive to shared, however.

If the user attempts to change access privileges, and for some reason is unable to get the new privileges, the old access privileges remain.

Multiple LU Considerations

The concept of exclusive access has been discussed in terms of multiple tasks sharing the same file. Here it was assumed that a single task does not attempt to assign the same file to multiple Logical Units (LU). Occasionally, however, a case arises when the task does desire to assign the same file to multiple LUs. This might occur as a result of default assignments or assignments made by the console operator. In this case, exclusivity applies between LUs just as between tasks. This means that a file connot be assigned for exclusive Read access on one LU and shared Read on another. If a file is assigned for exclusive Read or Write access on any given LU, it may not be assigned for that kind of access on any other LU.

Assume that a file is open on LU 1 and LU 2 at the same time. The following matrix shows the compatible combinations:

LU 1 \ LU 2	SREW	EWO	SWO	SRW	SRO	ERO	ERSW	ERW
SREW	–	–	–	–	*	–	–	–
EWO	–	–	–	–	*	*	–	–
SWO	–	–	*	*	*	*	*	–
SRW	–	–	*	*	*	–	–	–
SRO	*	*	*	*	*	–	–	–
ERO	–	*	*	–	–	–	–	–
ERSW	–	–	*	–	–	–	–	–
ERW	–	–	–	–	–	–	–	–

* = compatible
– = incompatible

ACCESS METHODS

OS/16 supports two methods of access to files. random and sequential. These methods may be intermixed without having to close and reassign the file. The chief mechanism used to implement these methods is the current record pointer.

The current record pointer is a number, ranging from zero to the number of records currently in the file, indicating the record to be read or written on the next sequential access to the file Each record is numbered in sequence, starting with ZERO

The current record pointer may be adjusted in one of several ways

1   It is set to ZERO by the following operations
       Rewind
       Backspace to filemark (except on contiguous files)
       Assigning (except for Write-Only access)

2.   It is set to the number of records in the file (the proper position to append new records) by the following operations.
       Assigning for Write-Only access
       Forward to filemark (except on Contiguous files)

3   It is decremented by one by a backspace record operation, unless the file is already positioned at its beginning

4   It is incremented by one as follows:
       Forward record (unless already at end of file)
       Sequential Read or Write

5   A random Read or Write sets the current record pointer to a value one greater than the record read or written.

Use of the Unbuffered Physical buffer management method causes the current record pointer to be reinterpreted to a certain extent This is discussed fully, later.

Random Access

For random access, the user supplies the record number to Read or Write. This record is found, data transfer is performed, and the current record pointer is set to point to the next sequential record. If the user continues to use random access, there is no need to pay attention to the current record pointer, since it is readjusted on every call However, the user may wish to read or write a sequence of records, starting with a known record number. In this case, one would use a single random call and follow with a number of sequential calls.

On Indexed files the user is somewhat restricted in the use of the random Write call. This call may be used to update any record currently in the file, or to append one record to the end of the file. If the record number specified is more than one record past the end of the file, the call is rejected with EOF (end of file) status. Effectively, this means that a file must be expanded sequentially. If the file has only five records, a sixth may be added but record number 100, for example, cannot. Otherwise, the system would have to create a large number of null, or vacant, records between the old end of the file and the new record For reasons of safety and time considerations, OS/16 does not do this.

On Contiguous files there is no restriction on the use of the random Write or Read call Any record within the file's allocation may be read or written

Sequential Access

Sequential access is the simplest and most common access method The user performs a series of sequential Read or Write calls. These cause records of the file to be read or written in sequence. The current record pointer is adjusted automatically at each access. The REWIND, FORWARD RECORD, BACKWARD RECORD, FORWARD FILE, and BACKWARD FILE commands may be used for repositioning as described previously.

BUFFER MANAGEMENT

OS/16 MT2 supports two buffer management methods. Buffered Logical (BL) and Unbuffered Physical (UP). One of these buffer management methods is specified each time a file is assigned. From that time until the file is next closed, the buffer management method may not be changed The buffer management method is logically independent of file structure, access method, or the contents of the file.

Buffered Logical

In the Buffered Logical (BL) method, files are divided into *logical records* These logical records may be of any length, regardless of the physical block size of the file. The logical record length for any given file is fixed at the time it is allocated. Thereafter, it is a permanent attribute of that file. It would be impossible therefore to write twenty-byte records on a file one day, and then the next day to write eighty-byte records on the same file without resorting to subterfuges.

It is possible to read or write less than a logical record; however. this can waste space, since the file is divided physically into logical records of the size specified at the time the file was allocated. Likewise, it is not possible to write variable-length records on a file without wasting space. In such a case, the logical record length specified at allocation time must be the size of the longest record ever to be written on that file. If the user tries to read or write a record that is longer than the file's logical record length, data is lost on a write operation, and is not returned on a read operation.

The BL buffer management method packs logical records into physical blocks as efficiently as possible, allowing logical records to overlap into the next physical block if necessary. For example, if the amount of space in a physical block that is available for data storage is 256 bytes (a common figure on an Indexed file), and the logical record length is 80 bytes, sixteen logical records are packed into five physical blocks. The logical record size may exceed the size of a physical block. The only restriction on logical record size is that no logical record may exceed 65,535 bytes.

All logical to physical transformations are handled automatically by the BL buffer management method. When a block is read or written, the actual data transfer takes place between the device and a buffer in system space. This buffer is not accessible to the user program When a task reads or writes a record, data is transferred between the task and the system buffer. Actual reads and writes take place only when required. All actions of the buffer management method are fully transparent to the user.

The BL buffer management method is probably the most convenient for general-purpose use. For sequential access, it is highly efficient, and it makes good use of disc space. Its primary drawback lies in the area of random Writes. In order to update a record, the physical block that the record lies in must be read first, to keep from destroying data in records that are in the same block but not being updated. Therefore, a random updating process may require a number of disc accesses that is far out of proportion to the number of records being updated.

Unbuffered Physical

The Unbuffered Physical (UP) management method does not operate on logical records, but works directly on physical blocks. Data is transferred directly from a buffer in the user program to the device, without being moved into a system buffer. For a Write operation, data is moved directly from the device to the user program.

The current record pointer is reinterpreted in the UP method In this buffer management method, the current record pointer points to the current physical block. Otherwise, it is the same as in the BL method. All data transfers must begin on a physical block boundary. The length of data to be transferred may be less than the size of a physical block. The Contiguous file structure permits data transfers larger than a physical block, of any size up to the total size of the file. In this file structure, therefore, the current record pointer may be incremented by more than one following a data transfer.

The advantages of the UP method lie in the speed with which data transfers can be done. There is no time required for moving data in core between system and user space if this method is used. Its primary disadvantage is that space on the volume is often wasted when using this method.

FILE STRUCTURES

OS/16 MT2 supports two file structures. Although these structures differ, in most cases the same data manipulations can be performed on one structure as another. The user's choice of file structure does not usually depend on the kind of data to be put in the file, but on the way the data is to be accessed. OS/16 file structures are each optimized for one specified form of access.

OS/16 file structures may be broadly divided into two categories: open-ended and closed. An open-ended structure may expand dynamically as new data is added. The maximum size of an open-ended file is limited only by the amount of space available on its volume. When an open-ended file is allocated, no space is reserved for it on the volume; space is only seized as it is required. The Indexed file structure is open-ended.

A closed structure is limited in size at the time it is allocated. Space is reserved for it at that time. Thereafter, it may neither expand nor contract. The Contiguous file structure is closed

The advantages of open-ended structures are many. First, the user need not worry about the maximum size of the file. (This parameter is often difficult or impossible to compute.) Second, unused space on the volume is not wasted, but is available for use by other files. This tends to permit a greater number of files per volume. In most cases, the user programmer should choose an open-ended file structure.

The primary advantage of closed structures lies in the fact that all the space required for the file is reserved at allocation time. If the maximum length of the file is known, the user may be sure that there is always room for the data and that no other tasks have preempted needed disc space. This advantage comes at the cost of wasted space on the volume.

At this time, not all access methods and buffer management methods are supported on all file structures.

Indexed Files

The only open-ended file structure in OS/16 MT2 Release 00 is the Indexed file. The Indexed structure consists of a chain of index blocks (see Figure 6-1). The first two fullwords of each index block are used as links. The first fullword points to the previous block (it is zero in the first block). The second fullword points to the next block (it is zero in the last block). The rest of the index block contains fullword pointers to the data blocks which are the physical records of the file. Two fullword pointers in the volume directory point to the first and last index blocks.

At allocation time, two physical block sizes may be specified. They are the index block size and the data block size Normally, these block sizes are one sector each (256 bytes), but any number of sectors may be specified. The index block size determines the number of data pointers to be contained in one index block. The data block size determines the amount of data space available in one data block. 256 bytes of data are contained in a one sector data block.
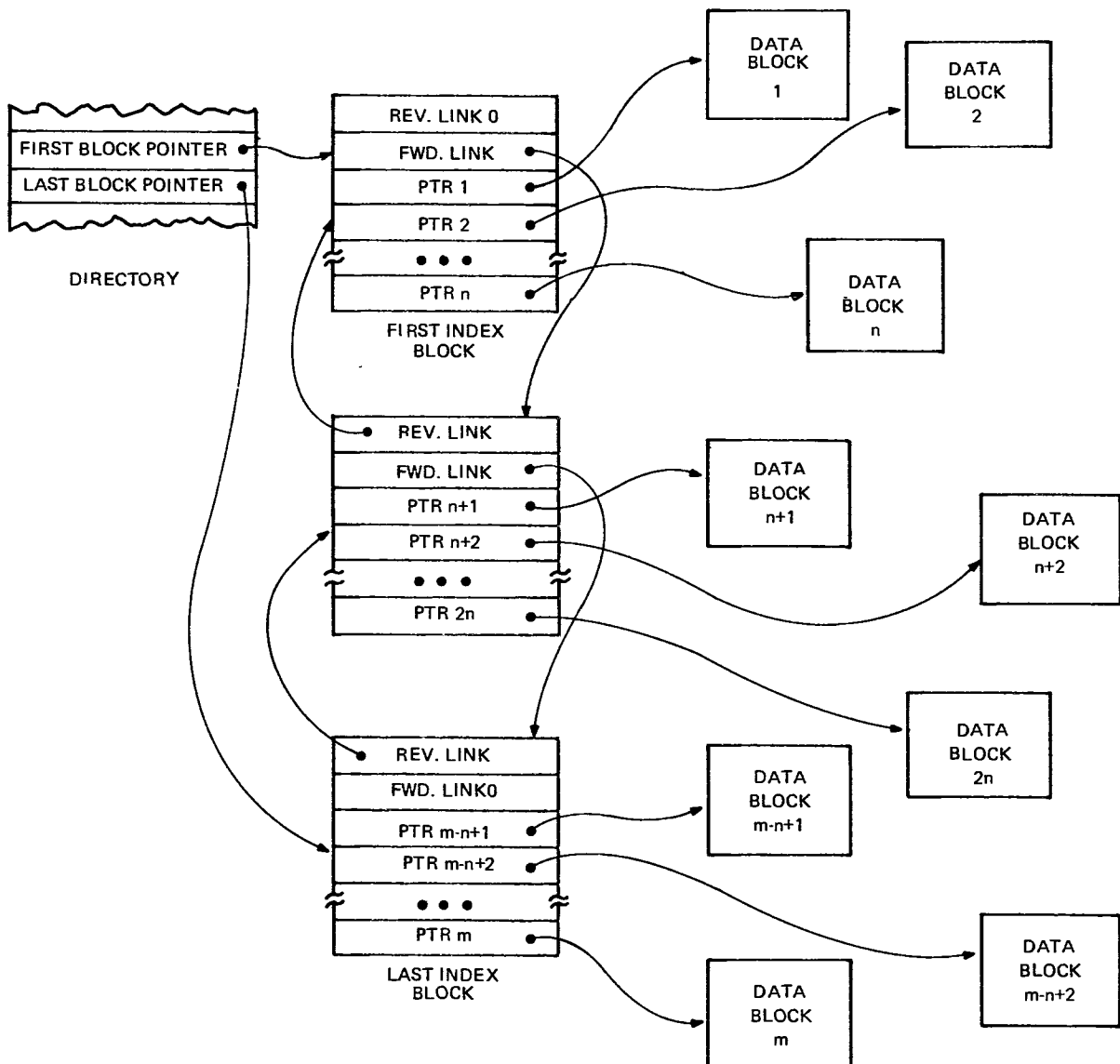
Figure 6-1. Indexed File Structure

Contiguous Files

The Contiguous file structure (see Figure 6-2) is a closed-ended file structure supported by OS/16 As the name implies, all blocks of a Contiguous file are allocated contiguously on the volume. The file size is specified at the time of allocation, and all required space is reserved at that time. This gives the Contiguous file certain characteristics that are unique to this file structure
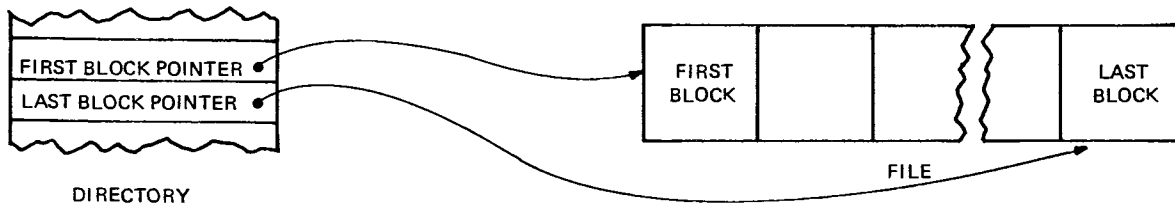
Figure 6-2. Contiguous File Structure

Random Reads and Writes may access any record or block on the file, regardless of which records have been previously written. This makes it possible to create a Contiguous file in a truly random fashion (At the same time, it should be noted that it is possible to read a record or block that has never been written. In such cases, the data returned is not defined.)

The Contiguous file supports a pseudo-filemark capability that gives it some of the characteristics of a magnetic tape device. The filemark is X'1313' at the beginning of a record or block. Care should be taken to ensure that this datum is not inadvertently written at the beginning of a record or block. The forward-file and backward-file operations on a Contiguous file work as they would on a magnetic tape. That is, the file is positioned forward or backward respectively until a "filemark" is found. The current record pointer is then left following or preceding this filemark, respectively. Note that this removes the capability of setting the file up for an append operation by using the forward-file command. (It may still be set up for append by initially opening it for Write-Only). The Write-filemark operation is implemented on this file type, and results in the writing of the X'1313' mentioned previously

It is permissible to read or to write data longer than the physical block length of a Contiguous file. The physical block length of a Contiguous file is always one sector, or 256 bytes. The current record pointer is adjusted to point to the next block following the data transfer, in this case.

Use of the Contiguous file provides the fastest possible means of access to Direct-Access storage. If time is of the essence, this structure should be used. However, it wastes space, is not open-ended, and requires a number of contiguous vacant sectors on the volume in order to allocate such a file. A Contiguous file, therefore, may be impossible to allocate on a heavily-used checker-boarded disc.

FILE MANAGEMENT

This section discusses the use of the Allocate, Assign, Close, Delete and Checkpoint calls The Rename and Reprotect calls have been discussed as has the Change Access Privileges call.

Allocation

When a file is allocated, its directory entry is built and, if it is a Contiguous file, space is reserved for it on the disc. The allocation of a file does not cause the file to be assigned; this is done by the Assign call. A file may be allocated either from the system console or from a user program. Regardless of how a file is allocated, however, certain information must be specified:

Volume ID:	This specifies the volume on which the file is to be allocated. It must be the name of an on-line Direct-Access volume; otherwise, "volume error" is returned.
Filename/extension:	This gives the newly allocated file its name. There must not be any other file of that name and extension on the specified volume; otherwise, "name error" is returned.
Write key/Read key.	This sets up the initial protection keys for the file.
Logical record length:	This sets the logical record length for the file. It may be any size up to 65,535 bytes; however, it may not be changed. If the logical record length is set to ZERO, the BL management method may never be used with this file.
Size	For a Contiguous file, this is the size of the entire file, in sectors. May be any size up to the maximum amount of contiguous space available on that volume at that time. If too large, "size error" is returned.
	For an Indexed file, this field is broken up into two halfwords. The first halfword is the index block size, the second halfword is the data block size. These block sizes are specified as a number of sectors (never greater than 255). If the index block size is specified as ZERO, an index block size of ONE is used.
File type	Indexed or Contiguous

Assignment

To assign a file to a Logical Unit, the Assign call (SVC 7) is used from a user program, or the ASSIGN command from the system console At this point, the desired access privileges and buffer management method must be specified. The Read key must be given, if Read access is requested; the Write key must be specified for Write access.

At the time of assignment, the system allocates, out of system space, a File Control Block (FCB) and buffers for the file. The buffer space required, if any, depends on the chosen buffer management method and on the file's physical block size If the physical block size of the file is too great for the amount of remaining system space, the file may not be assigned. In this case, a buffer error status is returned; the user program may attempt to relinquish some memory and then retry the assignment If that does not work, the user has little choice but to complain to the console operator and abort the task. Of course, if the user has another file assigned that can be closed, that can be done to release space to the system.

For this reason, the user should not keep files assigned longer than necessary. In addition, the physical block length of files should be kept as short as possible unless there are overriding considerations.

Closing

A file may be closed at any time by issuing a Close call No information other than Logical Unit need be specified. The system waits for any incomplete data transfers to terminate, flushes the buffers (that is, writes out to the volume any partly filled buffers), and updates the file's directory entry. The Logical Unit that the file was assigned to is deassigned.

Deletion

A file may be deleted only if it is not currently assigned to any task The user must supply the volume name, file name and extension, and both Read and Write keys. When a file is deleted, all its space on the volume is released and may be used by the files.

Checkpointing

The Checkpoint call performs the buffer-flushing and directory-updating functions of a Close call, without performing any of the other functions of a Close operation. This is essentially a protective operation that can be used to guard against system failure, either for very critical files, or for files assigned for lengthy periods of time without being closed

Since the directory is only updated at close or checkpoint time, data appended to a file after the latest close or checkpoint operation is lost at system failure for certain types of files and buffer management methods. When a file is closed or checkpointed, however, all data appended prior to that time is guaranteed safe through a system failure, providing that the system failure did not destroy the volume directory.

The checkpoint operation is never required for Contiguous files, since data is written directly to the disc, and this file structure does not use dynamic disc allocation techniques.

The checkpoint operation is not required for Indexed files if no new records are created (that is, if only already-existing blocks are being updated). If new records are being created (i.e., append operations), the checkpoint operation may be desirable

TRADEOFFS

In the areas of file structure, buffer management method, and access method, there are many choices which may be confusing to the programmer The following data is presented to assist the programmer in making this choice.

If pseudo-filemarks are required for magnetic tape emulation, the Contiguous file structure is required. This may be the case when magnetic tape-oriented programs are to be used.

If the maximum length of the file is completely unknown, the Contiguous file is ruled out. Only an open-ended file structure will do in this case. Most applications, however, generally define files in such a way that it is possible to make a good guess at the maximum length of each data structure. In this case, assuming the disc is not badly checkerboarded, the Contiguous file may be considered. (If the disc is checkerboarded, the allocation of a large Contiguous file may not be possible.)

Between the Indexed and Contiguous file structures, if the file is to be read and written using sequential access, the Indexed file structure is the obvious choice. If the file is long and is to be accessed randomly, the Contiguous file structure is usually preferable. However, this choice is not always clear, for the following reasons:

> Random access may be more or less efficient depending on the average access distance. If the distribution is equiprobable, that is, if the likelihood that any one record will be selected for the next data transfer is the same as for any other record, the mean search distance is 1/3 of the length of the file. If the distribution is highly localized, the Contiguous file structure has few advantages over the Indexed.

> If the REWIND and FORWARD-FILE commands (which do not cause disc accesses) are used, the mean search distance may be reduced from 1/3 of the file length to 1/6 of the file length. This also tends to make the Indexed file look a little better.

Once the choice of the Indexed structure has been made, the physical block size must be selected. Cogent reasons have been given for keeping the physical block size small. However, a large block size can be very helpful in some cases. The main time factors involved in a disc access are seek time (for moving-head discs) and rotational latency time. These times, on the average, far overshadow the actual data transfer time. Therefore, a transfer of two or more sectors generally costs little more time than a transfer of only one sector. For this reason, the number of disc accesses is the critical figure in computing the time required to access a file. A large physical block size can reduce by far the number of accesses. Here it is important to consider performance of the overall system. If a given task is not critical, and is running in a multitasking environment, other tasks can be active while it waits for disc accesses. If the task is critical, however, or is running in a single-tasking environment, a large physical block size may reduce its running time.

Finally, if speed of access is paramount and the file size is fixed, the Contiguous file structure should certainly be used. There is no time overhead associated with finding data using this file structure. The amount of system overhead used in accessing Contiguous files is less than for any other file type.

The compatibilities between these file types far outweigh their incompatibilities. It is quite possible to write programs that can use any or all of these file structures indiscriminately. If programs are written in such a manner, then the user can actually try out the application using the various possible file structures in order to find out which is more efficient. In addition, the programs are able to use files which may be the output of other dissimilar programs or which may previously have existed.

The Contiguous file is compatible with the Indexed file, providing that the Contiguous files extraordinary treatment of filemarks can be ignored.

CONCLUSIONS

The OS/16 file structures and access mechanisms are basic building blocks. They are not oriented towards any specific application, or toward any specific class of applications. These structures, methods and mechanisms are designed to give the user programmer a set of tools. By using these tools one should be able to build a set of data structures specifically suited to whatever the application may be.

CHAPTER 7
GUIDE TO USER OVERLAYS

INTRODUCTION

In order to minimize the amount of memory necessary to run a task, OS/16 MT2 allows the user to design a task in segments and control those segments such that only those that are necessary must be in memory at any one time. Such a task consists of the root segment, which is in memory for the duration of the task, and a number of overlay segments, which are brought into memory when necessary, at the task's request. This chapter discusses the use of overlays to reduce the memory required to execute a task.

A task to be overlayed usually is designed as a main program with a number of subroutines Before being executed under OS/16 MT2, this main program and its subroutines must be established as a task by the OS/16 Task Establisher Task (TET/16). TET/16 processes 16-Bit loader format programs and produces memory image load modules. TET/16 resolves EXTRN and ENTRY references by linking or editing in the specified programs. References to Task Common and Resident Library routines are also processed by TET (see Chapter 1 of this manual and Chapter 4 of the OS/16 MT2 Operators Manual). If no overlay facility is to be used, TET/16 processes the specified main program and subroutines and produces one image load module containing all the task's routines and data areas in the order processed. If overlays are to be used, TET/16 is used to produce an image load module containing the root segment and multiple areas of reserved space, and one or more image load modules that contain overlay segments to be loaded into the space reserved in the main program.

ROOT AND OVERLAY SEGMENTS

For an overlayed task, the root segment consists of all the programs, subroutines and data areas that must be in memory throughout the execution of the task. At TET time, the user must specify·

> The programs to be linked or edited into the root.
>
> All Block Data programs used by the task.
>
> The maximum amount of memory obtained through Get/Release Storage requests or any non-standard internal memory requirements.
>
> The starting address of the partition the task is being established for.

To produce overlay segments, the user specifies the programs and Block Data programs to be linked into each overlay and a unique name to be associated with the overlay load module. After all the overlays to be loaded into the first overlay area have been processed, the user can then specify the overlays to be loaded into a second overlay area. Up to 127 overlay areas can be defined in this manner. Overlays to be loaded into the same overlay area are said to be on the same level.

Figure 7-1 illustrates the organization of a sample memory map for a task using two overlay areas: the image load module containing the root segment and any common blocks necessary. This segment reserves space for Get Storage requests and for each overlay area. The user must specify the starting address of the partition the task is being established for. After processing the routines and data areas to be included in the root, TET reserves the user-specified number of bytes for Get Storage requests. If no value is specified, TET uses a value based on the requirements of the FORTRAN Run Time Library (RTL). In specifying the Get Storage area, the user must consider Get Storage requests issued by the root and all overlays, since all such requests are obtained from this area and any additional non-standard memory requirements of the task. From the size of the root segment, TET fixes the start of the first overlay area. The starting address of subsequent overlay areas is set to the starting address of the previous overlay area plus the size of the largest overlay produced for that previous area. The Get Storage area is reserved above the last overlay area
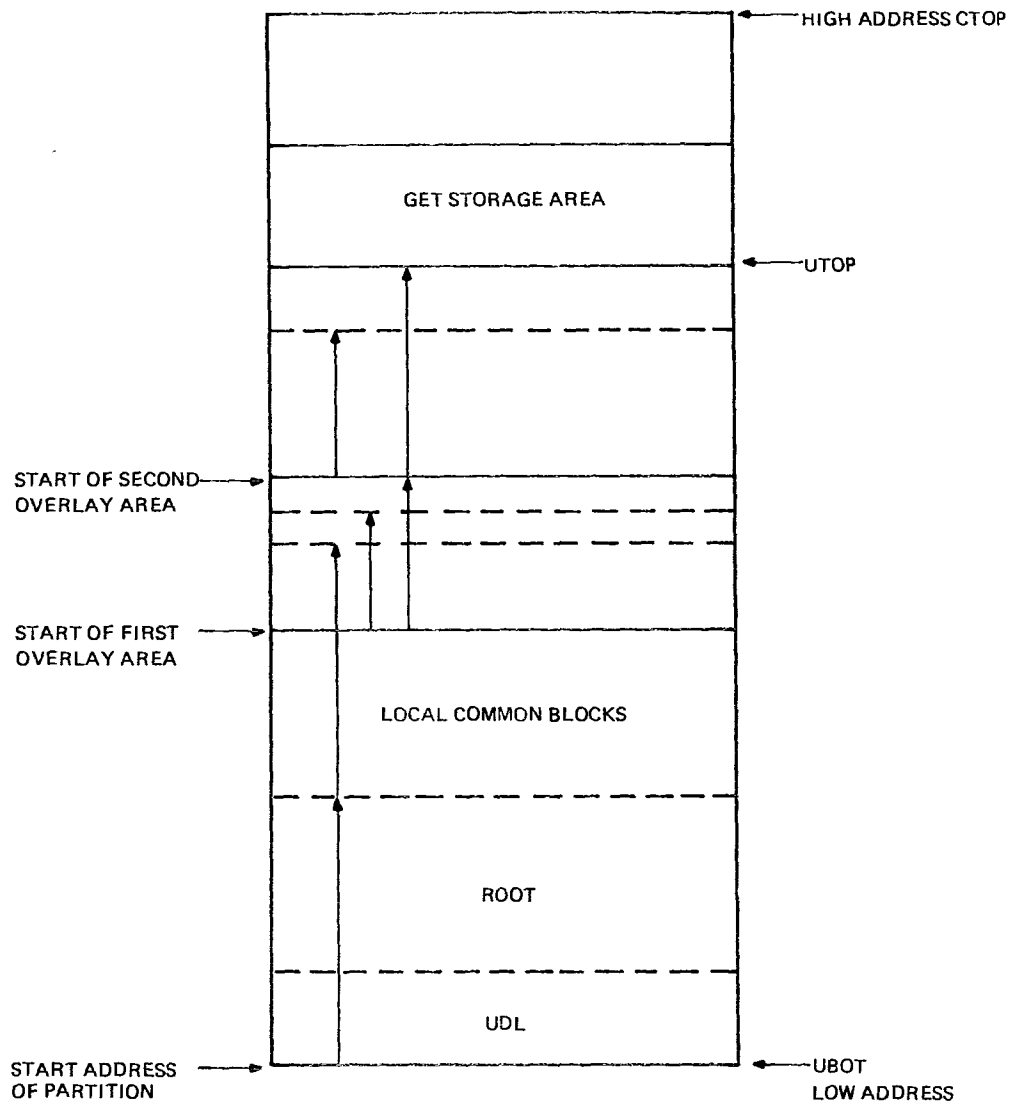
Figure 7-1. Partition For An Overlayed Task

OVERLAY CONVENTIONS

This section discusses the conventions that must be followed in the design of an overlayed task in order to ensure proper coordination between the various segments. At execution time, an overlay segment is loaded into memory via an SVC 5, Fetch Overlay request. The SVC 5 parameter block specifies the Logical Unit assigned to the file or device containing the desired overlay, and the name associated with the overlay at TET time An overlay load request may be issued by the root segment or by an overlay segment with the exception that an overlay must not be loaded in the overlay area occupied by the segment requesting the load. SVC 5 returns an error: if the overlay could not be loaded because of an I/O error, a name mismatch or the overlay does not fit into memory available (see Chapter 3 for SVC 5 details). The root can reference any overlay that is loaded and an overlay can reference the root or any overlay loaded into another overlay area. It is not necessary to load an overlay if it is known to be in memory.

In order to ensure proper resolution of EXTRN and ENTRY references no overlay may define an ENTRY already defined in the root or in an overlay in a different overlay area. Overlays designed for the same area may separately include the same subroutine name provided that there are no references to that name by segments not in that area. For example, suppose a task consisted of a root segment and three overlays sharing the same area. If two of the overlay segments require a common routine, such as the SINE subroutine, it is possible to link the SINE routine into each of the two overlays An error would result at TET time if the SINE routine was also linked into, or referenced by, the root.

OVERLAY DESIGN CONSIDERATIONS

Task Level

There are many considerations which determine the optimum number of overlay areas and what routines are loaded into the various areas This section suggests a few of these considerations.

If one large overlay is to share an area with a number of small overlays, determine if the large overlay could share another area with a similar size overlay, thus reducing the size of the first area. In other words, in the absence of coordination restrictions, a design which places only overlays of similar sizes in the same overlay area minimizes memory requirements. If all routines, referencing a particular common block, form one overlay, then the common block should be part of that overlay instead of requiring space in the root segment. If two overlays in the same overlay area communicate via a labeled Common block, then that block must be defined and positioned in the root segment or an overlay in another overlay area which is in memory during the time the Common block is referenced.

It may also be possible to use the Get Storage area for temporary storage used by Overlay Segments. If overlays are to use this area, it is important to coordinate Get/Release Storage (SVC 2) requests from the overlays with such requests from the root segment or from resident library routines. If overlays get the required storage on entry and release the same amount on exit, and all overlay segments are invoked as subroutines, the Get/Release requests would be processed in the correct order. In calculating the amount of Get Storage area to reserve, the user must consider the maximum number of bytes that may be gotten at any one time by the root segment, overlay segments and reentrant library routines.

System Level

In designing a series of tasks to be run as a system under OS/16 MT2, it is necessary to choose the most efficient partition configuration to contain the tasks. The same coordination considerations used in designing an overlay structure enter into designing the partitioning structure. The user must decide what tasks must be in memory simultaneously and what tasks may share a partition. After deciding which tasks may share a partition, these tasks should be established. The maps produced by TET can then be used to choose the starting address for the next partition. After all the partitions have been determined, the SET PARTITION operator command can be used to partition the user space.

OVERLAY EXAMPLE

Figure 7-1 illustrates the mapping of a partition containing a task with two overlay areas. The root segment consists of the User Dedicated Locations, the main program and subroutines, a number of local Common blocks defined in those subroutines and 512 bytes of Get Storage area. The first overlay area is designed to hold one of three overlays named ABC, DEF, or GHI. The second overlay area is designed to hold one of two overlays named JKL and MNO. Each overlay is to be output to a file named XXX.OVY where XXX is the overlay name. Overlay GHI consists of a main routine and several subroutines, the remaining overlays consist of a single subroutine Figure 7-2 illustrates the TET/16 commands used to establish the task.

ESTABLISH	TA	
INCLUDE	TASK.OBJ	LOAD MAIN PROGRAM
EDIT	SUBRTN.OBJ	LINK SUBROUTINES
GET	200	RESERVE 512 BYTES
OVERLAY	ABC,NEW	DEFINE NEW OVERLAY AREA FOR ABC
INCLUDE	ABC.OBJ	LOAD OVERLAY PROGRAM
OVERLAY	DEF	USE SAME AREA FOR DEF
INCLUDE	DEF.OBJ	LOAD OVERLAY PROGRAM
OVERLAY	GHI	USE SAME AREA FOR GHI
INCLUDE	GHI.OBJ	
EDIT	GHISBR.OBJ	LINK GHI SUBROUTINES
OVERLAY	JKL,NEW	DEFINE NEW AREA FOR JKL
INCLUDE	JKL.OBJ	LOAD OVERLAY PROGRAM
OVERLAY	MNO	USE SECOND AREA FOR MNO
INCLUDE	MNO.OBJ	
BUILD	TASK,TASK TSK	OUTPUT ROOT SEGMENT
BUILD	OV,ABC.TSK	AND OVERLAYS
BUILD	OV,DEF.TSK	
BUILD	OV,GHI.TSK	
BUILD	OV,JKL.TSK	
BUILD	OV,MNO.TSK	
MAP		OUTPUT MAP OF ROOT AND OVERLAYS
END		

Figure 7-2. Establish A Sample Overlayed Task

CHAPTER 8
INPUT/OUTPUT PROGRAMMING

INTRODUCTION

All input/output requests are made via Supervisor Call SVC 1. SVC 1 is described in general in Chapter 3. This chapter discusses the functional aspects of the devices and Direct-Access files supported by OS/16 MT2. Included is specific device dependent information such as supported functions, status returned, and formatting performed.

All devices and files support ASCII formatting, Proceed I/O, sequential access and non-unconditional proceed, unless otherwise noted in the individual driver or file handler description. The supported attributes listed with each description are in addition to those listed previously.

CONTIGUOUS FILES

Supported Devices

Contiguous files are supported on all devices supported by the Moving Head Disc Driver or the Floppy Disc Driver. Device Code is 0(0)

Supported Attributes

Read, Write, Binary, Wait, Random, Unconditional Proceed, Image, Rewind, Backspace Record, Forward Space Record, Write File Mark, Forward Space File Mark, Backspace File Mark. Fixed length 256 byte records.

Functional Description

Contiguous files may be accessed either randomly or sequentially. These two access methods may be mixed without closing and reassigning the file. Records are accessed by referring to the current record pointer. The current record pointer is a number, ranging from zero to one less than the number of sectors allocated to the file. For sequential access, the value of the current record pointer, as left by the previous operation, is used to access the file. For random access, the user specifies the current record pointer to be used. When a contiguous file is assigned, the current record pointer is set to zero.

Read:	(ASCII, Binary, and Image modes are identical.) The requested data is read from the disc, starting at the sector specified by the value of the current record pointer (sequential access) or by the random value specified in the SVC 1 call (random access), directly into the user specified buffer. The current sector pointer is set to 1 past the last sector involved in the transfer. All Reads begin on a sector boundary and end when the specified number of bytes have been transferred.
Write	Same as Read, except the data is transferred from the user specified buffer to the disc. If the number of bytes transferred is not an integral multiple of 256, the last byte in the buffer is propagated to fill the last sector involved in the transfer.
Test and Set:	Test and Set operation, specified by setting both Read and Write bits in the SVC 1 function code (X'60'), is implemented to provide compatibility with previous INTERDATA operating systems. Test and Set allows a task to access a file, or sector within a file, and at the same time mark it as being in use. Test and Set causes the requested data to be read into the specified buffer. Before control is returned to the task, the first halfword of the data is checked for a halfword of zero (X'0000'). If this halfword is zero, it is changed to X'FFFF', the record is rewritten to the file, and control is returned to the task with a Condition Code of zero. If the halfword is already X'FFFF', control is returned to the task with a Condition Code of X'F'. If the halfword is neither zero nor X'FFFF', control is returned with zero Condition Code. I/O proceed request for Test and Set (bit 4 of SVC 1 function code reset) is treated as a Wait request, except that an I/O Proceed complete queue entry is made, if enabled.

Rewind.	The current record pointer is set to zero.
Backspace Record	The current record pointer is decremented by one. If it is already zero, it is left at zero.
Forward Space Record.	The current record pointer is incremented by one. If it is equal to one less than the number of sectors in the file minus one, it is unchanged.
Write File Mark:	A record containing X'1313' as the first halfword is written to the current record The current record pointer is incremented by one.
Forward Space Filemark	The file is read from the current record until a record containing an X'1313' as the first halfword is found. The current record pointer is updated to point to the record following the file mark record. If no filemark record is found, the current record pointer is set to one less than the number of sectors in the file and End of Medium (EOM) status is returned.
Backspace File Mark.	The file is read backwards, starting with the record before the current record, until a record containing an X'1313' as the first halfword is found. The current record pointer is updated to point to the record containing the filemark. If no filemark record is found, the current record pointer is set to zero and End of Medium (EOM) status is returned.

Status Definition

Status	Meaning
X'00'	Normal Completion.
X'A0'	Device Unavailable.
X'90'	End of Medium (EOM). Request results in current record pointer equal to or greater than the number of sectors in file.
X'88'	End of File (EOF) Record containing X'1313' as first halfword detected during Read, Write, Forward Record or Backspace Record request.
X'84'	Unrecoverable error on device.
X'82'	Data Transfer error after 5 retries

NOTE

Contiguous files are truly random files. Records may be accessed in any order. The records are fixed length (256 bytes) and all sectors in the file are reserved in a contiguous block at allocate time Contiguous files support a pseudo filemark capability that gives them some of the characteristics of a magnetic tape device. Since the pseudo filemark is a halfword of X'1313' at the beginning of a sector, care should be taken to ensure that this datum is not inadvertently written at the beginning of a record.

INDEXED FILES

Supported Devices

Indexed files are supported on all devices supported by the Moving Head Disc Driver or the Floppy Disc Driver. Device code is 2(2)

Supported Attributes

Read, Write, Binary, Wait, Random, Unconditional Proceed, Image, Rewind, Backspace Record, Forward Space Record, Write File Mark, Forward Space File Mark, Backspace File Mark. Fixed length 256 byte records.

Functional Description

Indexed files may be accessed either randomly or sequentially These two access methods may be mixed without closing and reassigning the file. Records are accessed by referring to the current record pointer. The current record pointer is a number ranging from zero to one less than the number of records in the file. For sequential access, the value of the current record pointer, as left by the previous operation, is used to access the file. For random access, the user specifies the current record pointer to be used. When an Indexed file is assigned for Exclusive Write Only (EWO) or Shared Write Only (SWO), the current record pointer is set to the number of records in the file (the correct value for appending records); for all other access privileges, the current pointer is set to zero.

• Read:	(ASCII, Binary or Image modes are identical) The block containing the start of the logical record specified by the value of the current record pointer (sequential access) or by the random value specified in the SVC 1 parameter block (random access) is read into a system buffer (unless it is already in memory). The requested number of bytes are moved from the system buffer to the user specified buffer. The transfer ends on user buffer full or when a number of bytes equal to the logical record length of the file has been moved The current record pointer is set to the record following the record accessed. Any physical reads necessary to complete the transfer are completed by the system.
Write	Same as Read, except the data is moved from the user buffer to the system buffer. The system buffer is flushed, whenever necessary, by the system. If a current record pointer value of one greater than the last record in the file is specified, a record is appended to the file, thus allowing the Indexed file to be extended at any time. If the size of the specified buffer is less than the logical record length, the record is padded on the end with blanks (ASCII Format) or zeros (Binary Format).
Rewind	The current record pointer is set to zero.
Backspace Record·	The current record pointer is decremented by one. If already ZERO, it is left unchanged and EOF status is returned.
Forward Space Record.	The current record pointer is incremented by one If already equal to one less than the number of logical records in the file, it is left unchanged and EOF status is returned.
Forward Space File Mark·	The current record pointer is set to the number of logical records in the file.
Backspace File Mark	The current record pointer is set to zero.

Status Definition

Status	Meaning
X'00'	Normal completion.
X'90'	End of Medium (EOM). No space available on volume for new index or data block
X'88'	End of File (EOF) Attempt to set current record pointer to value greater than the number of records in file (write request only), to a value equal to or greater than the number of records in the file, or less than zero.
X'84'	Unrecoverable error during physical Read or Write.
X'82'	Data transfer error after 5 retries during physical Read or Write.

NOTE

Control is not returned to the task on I/O Proceed requests until completion of the request. Shared Write access is not permitted to an Index file. If a Shared Write access privilege (SRW, SWO, ERSW) is requested to an Indexed file, the system automatically grants exclusive Write access, if possible.

TELETYPE KEYBOARD/PRINTER

Supported Devices

Device	Product Number	Device Code
M33 Teletype Keyboard/Printer	M46-000/2/4/5	16(10)
M35 Teletype Keyboard/Printer	M46-001/3	17(11)
Non-edit CRT. current loop interface	M46-100/1	18(12)
Carousel 30/35 80 Char/line	M46-010/015/011/016	21(15)
Carousel 30/35 132 Char/line	M46-010/015/011/016 with M46-880	22(16)

Supported Attributes

Read, Wnte, Waıt, Unconditional Proceed, Image, Interactive, Halt I/O
Vanable Record Length up to 72 characters (16), 80 characters (17, 18, 21), 132 characters (22)

Functional Description

Read ASCII.   Data read is masked to 7-Bıt ASCII Dat.ı ıs read untıl the buffer is full, or a carriage return is found, whichever occurs first. Upon termınation, a Carriage Retum-Line Feed sequence is sent to the printer. Typing the character '#' causes the line input to be ignored, a Carrıage Return-Line Feed sequence to be output, and the Read operatıon to be restarted. Typing the character '←' or the BS character, causes the previous character entered to be ignored.

Wrıte ASCII.   The buffer is scanned to elımınate traıling blanks Data is then output untıl the buffer ıs exhausted or until a carriage return is found in the data stream. A line feed is automatıcally appended to the detected Carriage Return, or if no Carriage Return was detected, a Carriage Return-Lıne Feed sequence is output.

Read or Wrıte Image.   None of the above formattıng actıons occur. The amount of data requested ıs typed out or read ın, wıthout masking to 7-Bit ASCII, eliminating trailing blanks, checking for '#', '←' or BS characters, or detecting or appending carriage returns or lıne feeds. On Image read, however, a Carriage Return is detected as an end-of-line sentinel.

Status Definition

Status	Meanıng
X'00'	Normal Completion
X'82'	Time out or Lıne Break
X'84'	Unrecoverable Error
X'A0'	Device Unavailable

NOTE

The Reader/Punch of an ASR TTY is treated as a separate devıce but may not be actıve at the same tıme as the Keyboard/Printer.

TELETYPE READER/PUNCH

Supported Devices

Devıce	Product Number	Devıce Code
M33 TTY Reader/Punch	M46-000/2/4/5	81 (51)
M35 TTY Reader/Punch	M46-001/3	82 (52)
Carousel 35 Reader	M46-821	83 (53)

Supported Attrıbutes

Read, Write (except for Carousel Reader), Bınary, Wait, Uncondıtional Procéed, Image, Halt I/O
Vanable Length Records.

Functional Description

Read ASCII.   An X-ON character is output to turn the reader on Tape is read in blocked mode so data is not printed on thė printer whıle ıt is being read. Leadıng blank frames and Delete characters are ignored. Data is masked to 7-Bıt ASCII. The transfer is terminated on buffer full or Carriage Return, whichever occurs first. On termination of the transfer, tape ıs advanced to the next Delete character or blank frame. An X-OFF character is output to stop the tape

Read Binary:	An X-ON character is output to turn on the tape. Tape is skipped until the first non-blank frame is found. If the first non-blank character read is a X'F0', the following frames are read in until the user buffer is full. This is non-zoned binary format. If the first non-blank character read is not a X'F0', zoned binary mode is assumed. In this case, the characters are read, stripped of the zones and packed into the user buffer until buffer full In this mode the only valid punches are X'90', X'81'-X'84', X'95'-X'9F'. Other characters are ignored. On buffer full, the tape is advanced to the next blank frame in zoned binary mode Note that in unzoned binary, the first character transferred to the user buffer is the character following the X'F0' character, while in zoned binary, the first non-blank frame is transferred (after stripping and packing). After the request is satisfied, an X-OFF is output to turn off the reader.
Read Image	None of the above formatting operations is performed. An X-ON character is output to turn the tape on, data is read until user buffer full, the X-OFF character is output to turn the tape off and the transfer is complete
Write ASCII	The driver outputs a RUBOUT-TAPE-RUBOUT-RUBOUT sequence in order to initialize the TTY reperforator. Eight frames of blank tape are output as leader, the user data is output until buffer empty or Carriage Return, whichever occurs first. The driver ensures that a CR-LF-TAPE OFF-RUBOUT sequence terminates the record
Write Binary	The driver outputs a RUBOUT-TAPE-RUBOUT-RUBOUT sequence, followed by eight blank frames of leader. The user buffer is output, translating each byte into two frames of zoned binary data The transfer is terminated on buffer empty The driver then outputs a TAPE OFF-RUBOUT sequence.
Write Image:	None of the above formatting or control operations are performed. The user buffer is output until buffer empty.

Status Definition

Status	Meaning
X'00'	Normal completion
X'A0'	Device unavailable
X'84'	Data transfer error.
X'82'	Break detected during transfer. Timeout.

NOTE

On ASCII or Image write, it is possible to turn off the punch inadvertently by outputting a TAPE OFF character. On Image write it is the responsibility of the user to place the necessary control characters, such as TAPE and TAPE OFF in the user buffer to control the operation of the tape.

Since the Reader/Punch portion of the TTY is connected to the Keyboard/Printer portion, only one of these devices may be active at a time. On ASCII write, the data punched on the tape is also printed on the printer.

LOCAL CRT

Supported Devices

Device	Product Number	Device Code
Non Editing CRT (PALS or PASLA)	M46-100/1	34(22)
Graphic Display Terminal (PALS or PASLA)	M46-108/9	36(24)
Carousel 300 Keyboard/Printer (PALS or PASLA)	M46-803/4	37(25)
Carousel 300 Keyboard/Printer Electronic Format Control	M46-803/4 with M46-887	38(26)

All are supported via PALS or PASLA, M46-106 null modem (cable).

Supported Attributes

Read, Write, Wait, Unconditional Proceed, Image, Interactive, Halt I/O.
Variable length records up to 80 characters (34, 36), 132 characters (37, 38).

Functional Description

Read ASCII Data read is masked to 7-Bit ASCII. Data is read until the buffer is full, or a Carriage Return is found, whichever occurs first. Upon termination, a Carriage Return-Line Feed sequence is sent to the printer. Typing the character '#' causes the line input to be ignored, an LF/CR/CR sequence to be output, and the Read operation to be restarted. Typing the character "←", or the character BS, causes the previous character entered to be ignored.

Write ASCII The buffer is scanned to eliminate trailing blanks Data is then output until the buffer is exhausted or until a Carriage Return is found in the data stream. A Line Feed is automatically appended to the detected Carriage Return or if no Carriage Return were detected, a LF/CR/CR sequence is output.

Read or Write Image None of the above formatting actions occur. The amount of data requested is typed out or read in, without masking to 7-Bit ASCII, eliminating trailing blanks, checking for '#', '←', or BS characters, or detecting or appending Carriage Returns or Line Feeds. On Image read, however, as ASCII Carriage Return is detected as an end-of-line sentinel.

Status Definition

Status	Meaning
X 00'	Normal Completion
X'82'	Time out or Break.
X'84'	Unrecoverable I/O Error.
X'A0'	Device Unavailable.

HIGH SPEED PAPER TAPE READER/PUNCH

Supported Devices

Device	Product Number	Device Code
High Speed Paper Tape Reader/Punch	M46-242/3	80(50)

Supported Attributes

Read, Write, Binary, Wait, Unconditional Proceed, Image Variable record length.

Functional Description

Read ASCII. Leading blank tape and delete characters are ignored Data is masked to 7-Bit ASCII. Carriage Return terminates Read On termination, the tape is advanced until blank tape or a Delete character is read.

Read Binary: Tape is advanced until a non-zero character is read. If this character is X'F0', the tape is read until the buffer is full (Unzoned binary). If the first non-zero character is not X'F0', the tape is treated as a zoned binary tape. Each two characters are stripped of their zone and merged into one byte and placed in the buffer until buffer is full. On buffer full, the tape is advanced until blank tape is found. In zoned binary mode the only valid characters are: X'90', X'81'-X'84', X'95'-X'9F'. All other characters cause the transfer to end with unrecoverable status.

Read Image. Tape is read until buffer is full. None of the above formatting or control operations are performed.

Write ASCII Eight frames of blank tape are output. The user buffer is output up to (but not including) Carriage Return or until buffer empty. Carriage Return-Line Feed is then output.

Write Binary: Eight frames of blank tape followed by the character X'F0' are output. The user buffer is output until buffer empty.

Write Image The user buffer is output until buffer empty None of the above formatting or control operations are performed.

Status Definition

Status Returned	Meaning
X'00'	Normal Completion
X'A0'	Device Unavailable
X'82'	Time-out
X'84'	Transfer Error or invalid zone character

LINE PRINTER

Supported Devices

Device	Product Number	Device Code
Low Speed (200 LPM) Line Printer	M46-204/5/207/8	112(70)
High Speed (600 LPM) Line Printer	M46-209/10	114(72)

Supported Attributes

Write, Wait, Unconditional Proceed, Image Variable record length up to 132 bytes.

Functional Description

Write ASCII	The user buffer is output until a Carriage Return is found or until the buffer is empty. At the termination of the buffer, the system takes all necessary action to ensure that the buffer is printed and the paper spaced upwards one line. If form-feed or any other paper motion is desired, the appropriate characters must appear in the user's buffer.
Write Image:	The user buffer is output exactly as found in memory No action is taken by the system to ensure that the data is printed or that the paper is properly moved. The user should be familiar with the characteristics of the particular device being used.

Status Definition

Status	Meaning
X'00'	Normal Completion.
X'A0'	Device Unavailable Device not ready. Form error.
X'84'	Device became unavailable during transfer

NOTE

Although the Low and High Speed Line Printers have different form control characters, the sequence Carriage Return, X'01' causes a new line to be started on each printer. This sequence is used for Write ASCII to allow program compatibility using the same driver.

CARD READER

Supported Devices

Device	Product Number	Device Code
Card Reader Interface Without Translation	M46-235	96(60)
Card Reader Interface With Translation	M46-235 with M46-234	97(61)

Supported Attributes

Read, Wait, Unconditional Proceed, Image. Fixed record length 80 bytes (ASCII), 160 bytes (Image)

Functional Description

Read ASCII·	Each card column (12 bits) is converted into one 7-Bit ASCII character. Illegal codes are converted into the null character (X'00').

Read Image   Each column is converted into one halfword with the following format:

bit	0	1	2	3	4	5	6	7
card column	U	U	12	11	0	1	2	3

bit	8	9	10	11	12	13	14	15
card column	U	U	4	5	6	7	8	9

U = undefined

Status Definition

Status	Meaning
X'00'	Normal end of transfer.
X'A0'	Device unavailable. Reader not ready; Hopper empty or Stacker full
X'84'	Data Transfer error (Read check or Pick check)

MOVING HEAD DISC

Supported Devices

Device	Product Number	Device Code
2.5 Mbyte Removable Cartridge	M46-445/6	49(31)
5.0 Mbyte Fixed Cartridge	M46-440/1	50(32)
5.0 Mbyte Removable Cartridge	M46-440/1	51(33)
40 Mbyte Removable Cartridge	M46-429/30	52(34)
67 Mbyte Removable Cartridge	M46-600/2	53(35)
256 Mbyte Removable Cartridge	M46-604/6	54(36)

Supported Attributes

Read, Write, Binary, Wait, Random, Unconditional Proceed, Image Variable Length Records.

Functional Description

Read: A current sector pointer is maintained On a sequential read, data is read from the disc starting at the current sector into the user buffer until the buffer is full If an attempt is made to read beyond the end of the disc, end of medium status is returned. On a random request, the data is read from the disc starting at the sector specified by the random sector address passed with the request, until buffer full. ASCII, Binary, and Image requests are treated identically.

Write.   Data is written from the user buffer to the disc, starting at the current sector for sequential writes or at the specified sector for random writes, until buffer empty. Attempts to write past the end of the disc cause end of medium status to be returned. In this case, no data is transferred.

Errors on data transfers cause the operation to be retried ten times before returning error status.

All data transfers start on a sector boundary but may end on any byte of a sector.

Status Definition

Status	Meaning
X'00'	Normal completion.
X'A0'	Request could not be started, device not ready.
X'90'	End of medium. Transfer ends beyond the end of the disc.
X'88'	Pseudo end of filemark read. This status is only returned on request by the OS/16 File Manager.
X'84'	Seek incomplete. All device errors other than data transfer error.
X'82'	Data transfer error after ten retries. Write protection violation. Timeout.

8-8

The Moving Head Disc Driver is designed for use by the OS/16
MT2 File Manager. The Disc Driver cannot be invoked by a user
program unless it is an E-task. For user tasks, the disc is acces-
sed by means of the Contiguous or Indexed file handlers dis-
cussed in the first section of this chapter.

The 10 Mbyte Disc System, Product Number M46-440/1, is
supported in OS/16 as two 5.0 Mbyte cartridges, one fixed and
one removable. Since there is only one seek mechanism, the
two discs on a 10 Mbyte system share a busy flag and only one
disc may be active at any instant in time

FLOPPY DISC

Supported Devices

Device	Product Number	Device Code
Floppy Disc Drive	M46- (TBD)	55(37)

Supported Attributes

Read, Write, Binary, Wait, Random, Unconditional Proceed, Variable Length Records

Function Description

Read:
A current sector pointer is maintained. On a sequential read, data is read from the disc starting at
the current sector into the user buffer until the buffer is full. On a random request, the data is read
from the disc starting at the sector specified by the random sector address passed with the request,
until the buffer is full. If an attempt is made to read beyond the end of the disc, end of medium
status is returned with no data transferred. ASCII, Binary, and Image requests are treated identi-
cally

Write.
Data is written from the user buffer to the disc, starting at the current sector pointer (for sequential
writes) or at the specified sector (for random writes), until buffer empty. If an attempt is made to
write beyond the end of the disc, end of medium status is returned with no data transferred ASCII,
Binary, and Image requests are treated identically.

Errors on data transfers cause the operation to be retried ten times before returning error status.

All data transfers start on a logical 256 byte sector boundary (two physical sectors on the disc). A transfer may end on any
byte of a sector.

Status Definition

X'00'	Normal completion.
X'A0'	Request could not be started, device not ready.
X'90'	End of medium. Transfer ends beyond the end of the disc.
X'88'	Pseudo end of filemark read. This status is only returned on request by the OS/16 File Manager.
X'84'	All device errors other than data transfer error
X'82'	Data transfer error after ten retries. Write protection violation or Time-out.

The Floppy Disc Driver is designed for use by the OS/16 File
Manager. The Driver cannot be invoked by a user program unless it
is an E-task. For user tasks, the disc is accessed by means of the
Contiguous or Indexed file handlers discussed in the first section
of this chapter.

CASSETTE

Supported Devices

Device	Product Number	Device Code
Intertape Cassette Drive	M46-400	66(42)

Supported Attributes

Read, Write, Binary, Wait, Unconditional Proceed, Image, Rewind, Backspace Record, Forward Space Record, Write File Mark, Forward Space File Mark, Backspace File Mark Variable length Records.

Functional Description

Read
(ASCII, Binary, and Image modes are identical) Data is read from the cassette into the user's buffer The transfer terminates at buffer full, or at end of record, whichever comes first. If the record is longer than the buffer, error status is not returned. Parity errors in the unread part of the record, however, may be detected. If a parity error occurs, five retries are attempted before error status is returned. On parity error status return, the tape is positioned in the inter-record gap following the errored record.

Write.
(ASCII, Binary, and Image modes are identical.) Data is written from the user's buffer until the buffer is empty. The system retries five times on parity errors.

Status Definition

Status	Meaning
X'00'	Normal completion.
X'C0'	Attempt to transfer less than 4 bytes.
X'A0'	Device not ready Tape failed to move at start of request.
X'90'	End of tape on Read, Write or Write File Mark.
X'88'	End of file
X'84'	Device became unavailable during a request.
X'82'	Data transfer error after 5 retries; time-out.

NOTE

The driver cannot distinguish between Beginning Of Tape (BOT) and End Of Tape (EOT) conditions. Either condition generates an End Tape (ET) condition. If ET is detected on a request, with no previous request which causes forward motion, then BOT is assumed. The user should ensure that a cassette is not loaded at End Of Tape unless that condition is expected

Since the two drives on an Intertape Cassette share logic, a shared busy condition exists and only one drive of a cassette pair (e.g..X'45' and X'55') may be active at a time. Also Keys and Access Privileges are shared so the two drives may not be assigned with uncompatible privileges.

Continuous mode operations are used where allowed when requests are passed to the driver within the time required (10-milliseconds for Read, 30 milliseconds for Backspace).

9 TRACK MAGNETIC TAPE

Supported Devices

Device	Product Number	Device Code
9 Track 800 Bpi Magnetic Tape Interface	M46-470	64(40)
9 Track 1600 Bpi Magnetic Tape Interface	M46-475	65(41)

Supported Attributes

Read, Write, Binary, Wait, Unconditional Proceed, Image.
Rewind, Backspace Record, Forward Space Record, Write File Mark, Forward Space File Mark,
Backspace File Mark, Variable Length Records

Functional Description

Read:	Data is read into the user buffer from the magnetic tape The transfer ends on buffer full or end of record, whichever comes first. If the record is longer than the user buffer, error status is returned. This error status is indistinguishable from true parity error. On a parity error, five retries are attempted before error status is returned. After a parity error the tape is positioned in the inter-record gap following the errored record.
Write·	Data is written from the user buffer to the magnetic tape until the buffer is empty. Five retries are attempted on parity error.

For both Read and Write requests, ASCII, Binary, and Image requests are identical.

Status Definition

Status	Meaning
X'00'	Normal completion.
X'A0'	Device not ready. Tape unavailable at the start of request for data transfer.
X'90'	End of tape. Request caused the reflective marker at end or beginning of tape to be sensed.
X'88'	End of file. Filemark detected during request.
X'84'	Device became unavailable during request.
X'82'	Data transfer error after 5 retries. Timeout occurs.

NOTE

The driver cannot distinguish between Beginning Of Tape (BOT) and End Of Tape (EOT) conditions. Either condition generates an end tape (ET) condition. The driver assumes that if ET is detected on a Write request that the tape is at BOT. Issuing a Write request at EOT will hang up the controller such that no further interrupts may be serviced. If rewind is issued at BOT, the driver simply returns normal status. The user should ensure that the tape is loaded at BOT unless some other condition is expected. Backspace File Mark to a tape with no filemarks between the position of the tape and the point of BOT also hangs up the controller. The Processor Initialize key must be used to clear this condition if no clock exists to time out the driver

7-TRACK MAGNETIC TAPE

Supported Devices

Device	Product Number	Device Code
7 Track 800 Bpi Magnetic tape interface	M46-474	67(43)

Supported Attributes

Read, Write, Binary, Wait, Unconditional Proceed, Image Rewind, Backspace Record, Forward space. Record, Write File Mark, Forward Space File Mark, Backspace File Mark. Variable Length Records.

Functional Description

Read·	Data is read into the system buffer from the magnetic tape. This buffer is 342 bytes long. The transfer ends on buffer full or end of record, whichever comes first. If the record is longer than the system buffer, error status is returned. This error status is indistinguishable from the parity error. On a parity error, five retries are attempted before error status is returned. After a parity error the tape is positioned in the inter-record gap following the errored record.
	If no error occurs, the record is then formatted as follows:
Read ASCII:	Each byte from the system buffer is converted to 7-bit ASCII and stored in the user buffer. This conversion continues until end of user buffer or end of record, whichever comes first

Read Binary	Every 4 bytes from the system buffer are converted to three 8-bit binary bytes and are stored in the user buffer. This conversion continues until end of user buffer or end of record, whichever comes first
Read Image	Data is read directly from the magnetic tape into the user buffer None of the previously described conversions take place.
Write ASCII	Each byte from the user buffer is converted to 6-bit CDC compatible 7 track code Bytes other than carriage return X'0D' outside the range X'20'-X'5F' are converted to blanks and error status is not returned If the size of the user buffer is larger than 342 bytes, Illegal function status is returned. After the described conversion, the system buffer is written to the tape Five retries are attempted on parity error.
Write Binary	Every 3 bytes from the user buffer are converted to four 6-bit bytes and are stored in the system buffer. If the size of the user buffer is larger than 256 bytes, Illegal function status is returned After the described conversion, the system buffer is written to the tape. Five retries are attempted on parity error
Write Image	Data is written directly from the user buffer onto the magnetic tape. All data bytes are masked to 6 bits by the device. Five retries are attempted on parity error.

Status Definition

Status	Meaning
X'00'	No Errors
X'C0'	Illegal Function The user's buffer is too long with respect to the length of the system buffer
X'A0'	Device Unavailable The magnetic tape unit is off-line.
X'90'	End-of-Medium — The end of tape or beginning of tape markers have been detected.
X'88'	End-of-File — A Filemark has been detected
X'84'	Unrecoverable Error — Device became unavailable during request.
X'82'	Recoverable Error — Data transfer error after 5 retries. Timeout occurs.

EIGHT LINE INTERRUPT MODULE

Supported Devices

Device	Product Number	Device Code
Eight Line Interrupt Module	M48-001	128(80)

Supported Attributes

SINT

Functional Description

The Eight Line Interrupt Module provides eight interrupt lines to the processor from external equipment and acknowledges interrupts on a priority basis. Any line may be selectively enabled or disabled Several lines may be concurrently enabled An interrupt does not transfer any data, nor is any status given. For details on the functions supported, see the SVC 6 description in Chapter 3.

Status Definition

See SVC 6 error codes in Chapter 3

DIGITAL MULTIPLEXOR

Supported Devices

Device	Product Number	Device Code
Digital Multiplexor	M07-860	129(81)

Supported Attributes

Read, Write, Wait, Unconditional Proceed, Image, Binary, Random.

Functional Description

Read: The second byte of the random address field contains the segment and point number to be read. Data is read from the point specified until the buffer specified by the starting and ending address is full.

Write The second byte of the random address field contains the segment and point number to be written to. Data is written until the buffer specified by the starting and ending address is exhausted.

Status Definition

Status	Meaning
X'00'	Normal completion
X'A0'	Device unavailable

CHAPTER 9
GUIDE TO WRITING DRIVERS

INTRODUCTION

Drivers for non-standard devices may be readily included in OS/16 MT2 provided these drivers are written to conform with those procedures used in writing standard drivers. Standard drivers are designed to minimize the period of time external interrupts are disabled. To this end, all drivers consist of at least two, and sometimes three or four distinct parts. The first part or module required in all drivers, is the initialize routine. In operation, this part of the driver becomes a subroutine of the Executive, which runs with external interrupts enabled. The initialize routine interprets the function code, and in general, performs any preliminary function that can be done with interrupts enabled. The second module of the driver is the interrupt service routine. This is an independent routine, although assembled along with other parts of the driver. The third module which, depending on the type of device and the needs of the user, may not be required, is the termination routine. Like the initialize routine, It becomes a subroutine of the executive and performs such functions as error identification, code conversion, and interpretation of final device status. The fourth module is the abort routine, which stops on-going I/O when a cancel request is made.

The DEVICE CONTROL BLOCK

A fifth module, not technically part of the driver, is required for all devices on the system. This is the Device Control Block (DCB). One DCB is required for each device on the system. However, if the driver is properly coded, only one copy of the driver is needed to control several devices of the same type. The Device Control Block provides external storage locations for the driver. Thus, it is possible, using indexed instructions when referring to the DCB, for the driver to handle more than one device. Drivers and system routines reference locations in the DCB with indexed instructions. General Register 14 should always be used as the index register to the DCB. A DCB for each device including user-written driver devices, is automatically generated by the Configuration Utility Program (CUP/16), when the device is configured.

A map of the standard DCB is shown in Figure 9-1. The symbol 'DBxxxx' is the address of the DCB. Reference to elements of the DCB can be made symbolically from the DCB address. The STRUC in Figure 9-2 defined DCB elements. For example, the device code at DCB-18 may be referred to by DCB.DCOD(R14). The following section discusses DCB elements as pertains to user-written drivers.

DCB RECL

DCB.CNTS

DCB ATRB

DCB.KEYS

DCB BUSY

DCB.TERM

DCB ALOC

DCB.TOUT

DCB.FLGS

DCB INIT

DCB OPSW

DCB.NPSS

DCB.ENTR

DCB LEAV

DCB.NOPI

DCB.RSAV

DCB.ISR

DCB.RES

Offset		Content
-22		MAXIMUM RECORD LENGTH
-20	DCB WCNT WRITE COUNT	DCB.RCNT READ COUNT
-18	DCB.DCOD DEVICE CODE	DCB.DN DEVICE NO.
-16		ATTRIBUTES
-14	DCB.WKEY WRITE KEY	DCB.RKEY READ KEY
-12	DCB.SBSY	A(BUSY FLAG)
-10		A(TERM ROUTINE)
-8		A(ABORT ROUTINE)
-6		TIME-OUT COUNT
-4		FLAGS
-2		A(INITIALIZATION ROUTINE)
0(0)		OLD PSW
4(4)		NEW PSW STATUS
6(6)		STM R8,IORSAV
		LM R8,REG8xxxx
		BR R15
		STM R8,REG8xxxx
		LM R8,IORSAV
		LPSW DBxxxx
28(1C) R2	CALLER TCB POINTER AND I/O WAIT POINTER	DCB.IOW
30(1E) R3		A(PARBLK)
32(20) R4		FUNCTION CODE AND LU
34(22) R5		MAY BE USED BY DRIVER
36(24) R6		DEVICE NUMBER
38(26) R7		STATUS
40(28)		R8-R13
52(34) R14		A(DCB)
54(36) R15		A(ISR)
56(38)		BUSY FLAG

DBxxxx

Figure 9-1. Device Control Block Map

DCB.RECL	EQU	-22	LOGICAL RECORD LENGTH	-2B
DCB.CNTS	EQU	-20	COUNTS	-2B
DCB.WCNT	EQU	-20	WRITE COUNT	-1B
DCB.RCNT	EQU	-19	READ COUNT	-1B
DCB DCOD	EQU	-18	DEVICE CODE	-1B
DCB.DN	EQU	-17	DEVICE #	-1B
DCB.ATRB	EQU	-16	DEVICE ATTRIBUTES	-2B
DCB.KEYS	EQU	-14	KEYS	-2B
DCB.WKEY	EQU	-14	WRITE KEY	-1B
DCB RKEY	EQU	-13	READ KEY	-1B
DCB.BUSY	EQU	-12	A (BUSY FLAG)	-2B
DCB.SBSY	EQU	-12	A (SELCH BUSY FLAG)	-2B
DCB.TERM	EQU	-10	A (TERMINATION ROUTINE)	-2B
DCB.ALOC	EQU	- 8	ABORT LOC	-2B
DCB.TOUT	EQU	- 6	TIME OUT COUNT·0=TIMED OUT	-2B
DCB.FLGS	EQU	- 4	FLAGS HALFWORD	-2B
DCB.INIT	EQU	- 2	A (INITIALIZATION ROUTINE)	-2B
	STRUC			
DCB.OPSW	DS	4	OLD PSW SAVE AREA	
DCB.NPSS	DS	2	NEW PSW STATUS	
DCB.ENTR	DS	10	INTERRUPT SERVICE ENTRY POINT	
DCB.LEAV	DS	4	INTERRUPT SERVICE EXIT LOCATION	
DCB.NOPI	DS	8	NOP ISR ENTRY	
DCB.IOW	EQU	*+1	I/O WAIT POINTER	-1B
DCB.RSAV	DS	14*2	SAVE AREA FOR REGISTERS 2-15	
DCB.ISR	EQU	*-2	ISR POINTER (R15)	-2B
DCB.RES	DS	2	RESERVED FOR BUSY FLAG	

FLAGS MASKS

DFLG.BLM	EQU	X'8000'	BULK DEVICE FLAG
DFLG.LNM	EQU	X'4000'	ON-LINE FLAG
DFLG.BBM	EQU	X'2000'	BISYNC BUSY FLAG
DFLG.MPM	EQU	X'1000'	BIT MAP PRESENCE
DFLG.PFM	EQU	X'0800'	PSEUDO FILEMARK CHECK
DFLG.BMM	EQU	X'0400'	BIT MAP MODIFY FLAG
DFLG.CNM	EQU	X'0200'	CONSOLE FLAG
DFLG.BIM	EQU	X'0100'	BISYNC DEVICE FLAG
DFLG.S6M	EQU	X'0080'	SVC 6 CONNECTABLE DEVICE
DFLG.WPM	EQU	X'0040'	WRITE PROTECT FLAG
DFLG.NSM	EQU	X'0020'	SET=NO SEEK DONE,RESET=SEEK DONE
DFLG.S1M	EQU	X'0010'	FMGR CALL FROM SVC 1 FLAG
DFLG.S0M	EQU	X'0008'	START AT ZERO FLAG (GETSEC)
DFLG.IUM	EQU	X'0004'	DCB IN USE FLAG

ATTRIBUTES MASKS

DATR.INM	EQU	X'8000'	INTERACTIVE DEVICE
DATR.RDM	EQU	X'4000'	SUPPORTS READ
DATR WRM	EQU	X'2000'	SUPPORTS WRITE
DATR.BIM	EQU	X'1000'	SUPPORTS BINARY
DATR.WTM	EQU	X'0800'	SUPPORTS WAIT I/O
DATR.RNM	EQU	X'0400'	SUPPORTS RANDOM
DATR.UPM	EQU	X'0200'	SUPPORTS UNCONDITIONAL PROCEED
DATR.IMM	EQU	X'0100'	SUPPORTS IMAGE
DATR.HIM	EQU	X'0080'	SUPPORTS HALT I/O
DATR.RWM	EQU	X'0040'	SUPPORTS REWIND
DATR.BRM	EQU	X'0020'	SUPPORTS BACKSPACE RECORD
DATR.FRM	EQU	X'0010'	SUPPORTS FORWARD SPACE RECORD
DATR.WFM	EQU	X'0008'	SUPPORTS WRITE FILEMARK
DATR.FFM	EQU	X'0004'	SUPPORTS FORWARD SPACE FILEMARK
DATR.BFM	EQU	X'0002'	SUPPORTS BACKSPACE FILEMARK
DCOD.NUL	EQU	255	DEVICE CODE OF NULL DEVICE
	ENDS		

.

Figure 9-2. DCB — Device Control Block

Maximum Record Length

For user-written drivers, this field is zero since no device record length checking is performed.

Read/Write Count

This field must not be used by the driver.

Device Code

This field contains one of the valid user-written driver device codes (250-254) supplied to CUP/16 with the DEVICE statement.

Device Number

This field contains the physical device address supplied to CUP/16 with the DEVICE statement

Attributes

This field contains the value X'7F7E' for user-written devices — all operations are supported

Bit #	Meaning if set
0	Interactive Device
1	Supports read
2	Supports write
3	Supports binary
4	Supports wait I/O
5	Supports random
6	Supports unconditional proceed
7	Supports image
8	Supports Halt I/O
9	Supports rewind
10	Supports backspace record
11	Supports forward space record
12	Supports write file mark
13	Supports forward space filemark
14	Supports backspace filemark

Keys

This field is initially zero but may be modified by SVC 7 (Write key/Read key — 1 byte each).
This field must not be used by the driver

A (Busy Flag)

This field contains a pointer to the device busy flag.

A (Abort Termination Routine)

This field contains the address of the stop I/O routine (UxxTRM) used by the Executive in terminating I/O in progress.

Time-out Count

This field, initially zero, may be altered by the driver if time-out service is required. The time-out count in seconds is set by the driver when the I/O is initiated and decremented by the clock driver When the value is zero, the clock driver executes a simulate interrupt instruction on the device, notifying the driver of the device time-out.

Flags

This field is initially set to X'4000' indicating that the device is on-line. Other bits in the flags halfword do not apply to user-written devices.

A(Initialization Routine)

This field contains the address of the driver initialization routine (UxxDVR).

Old PSW

The next locations in the DCB are:

```
DBxxxx      DC      0,0         OLD PSW STATUS AND LOC
            DC      ISSTAT      NEW STATUS
```

DBxxxx is the address placed in the Interrupt Service Pointer Table. It points to an old Program Status Word save location where the current PSW is stored on an immediate interrupt.

New Status

The next location is the new status for an immediate interrupt. This status is X'3000' (ISSTAT), machine malfunction and fixed point divide fault interrupts enabled, all others disabled. When the immediate interrupt occurs, the Location Counter is forced to the next location following the new status:

```
STM     R8,IORSAV       SAVE INTERRUPTED REGISTERS
LM      R8,REG8xxxx     LOAD ISR REGISTERS
BR      RF              BRANCH TO INTERRUPT SERVICE ROUTINE
STM     R8,REG8xxxx     SAVE ISR REGISTERS
LM      R8,IORSAV       LOAD INTERRUPT REGISTERS-DCB.NOPI
LPSW    DBxxxx          RETURN TO INTERRUPT LOCATION
```

xxxx is the device mnemonic supplied to CUP/16 on the DEVICE statement

In this block of code, the registers in use at the time of the interrupt are saved in IORSAV, a common save area in the Executive. Eight registers must be saved, even if the interrupt service routine does not use that many. This means that the interrupt service routine can only use Registers 8 through 15. REG8xxxx is a location within the DCB

Register Save Area

The next locations are the register save area.

```
DC      0   R2      CALLER TCB POINTER AND I/O WAIT POINTER
DC      0   R3      ADDRESS OF PARAMETER BLOCK
DC      0   R4      FUNCTION CODE AND LU
DC      0   R5
DC      0   R6      DEVICE NUMBER
DC      0   R7      STATUS REGISTER, INITIALLY ZERO
DC      0   R8
        .   .
        .   .
        .   .
DC      0   RE      ADDR OF DCB
DC      0   RF      ADDRESS OF INTERRUPT SERVICE ROUTINE
```

It is the responsibility of the driver initialize routine to set up this area. Registers 2, 3, 4, 6, and 7 are loaded by the Executive, but it is up to the driver initialize routine to store them and any other registers it may load for the interrupt service routine. Register 15 is initialized by System Initialization to contain the address of DCB.NOPI. This causes interrupts to be ignored until Register 15 is set up by the driver initialization routine.

Busy Flag

The next location in the DCB is the busy flag. For devices that share a busy flag, the user may modify A(BUSY FLAG) in the DCB to point to a common busy flag. This is done by modifying the output of CUP/16 using OS EDIT.

If a Selector Channel (SELCH) is to be used, then A(BUSY FLAG) should be modified to point to a SELCH busy flag The SELCH busy flags generated by the system are of the form FFxxBSY where xx is the device number of the SELCH The following statements appear at the end of the DCB·

```
ORGxxxx     ORG       X'nn'*2+ISPTAB
            DC        A(DBxxxx)
            ORG       ORGxxxx
```

In these statements, X'nn' is the device number, and the ORG statement sets the Location Counter of the Assembler to the location in the Interrupt Service Pointer Table where the address of the DCB must be stored The second ORG sets it back to its previous location. For SELCH devices, the driver itself must set up the correct slot in the Interrupt Service Pointer Table, i.e, STH R14,X'F0'*2+X'D0'.

Device codes 240-254 are reserved for user-written drivers For example, the CUP/16 statement DEVICE 240,3A,USER: generates the following DCB·

```
          EXTRN     U40TRM,U40DVR     DRIVER ABORT ROUTINE AND DRIVER ADDRESS
          ENTRY     DBUSER            DCB ADDRESS
          DC        X'0000',0         MAX RECORD LENGTH, READ/WRITE COUNTS
          DC        X'F03A'           DEVICE CODE, DEVICE NUMBER
          DC        X'7F7E',0         ATTRIBUTES, KEYS
          DC        USERBSY,0,U40TRM,0 ADDRESS OF BUSY FLAG, TERM AND
                                       ABORT ROUTINES, TIMEOUT COUNT
          DC        X'4000',U40DVR    FLAGS, ADDRESS OF DRIVER
DBUSER    DC        0,0,ISSTAT        OLD PSW, NEW PSW STATUS
          STM       R8,IORSAV         SAVE REGISTERS ON INTERRUPT
          LM        R8,REG8USER       LOAD DRIVER REGISTERS
          BR        RF                GO TO DRIVER
          STM       R8,REG8USER       SAVE DRIVER REGISTERS
          LM        R8,IORSAV         LOAD INTERRUPTED REGISTERS
          LPSW      DBUSER            RETURN
REG2USER  DC        0,0,0,0,0,0       REGISTER SAVE AREA
REG8USER  DC        0,0,0,0,0,0,0,0
USERBSY   DC        X'00'             BUSY FLAG
ORGUSER   ORG       X'3A'*2+ISPTAB    ORG TO ISP TABLE SLOT
          DC        DBUSER            DEPOSIT DCB ADDRESS
          ORG       ORGUSER           RE-ORG
```

NOTE

The abort and driver initialization routine addresses are always of the form UxxTRM and UxxDVR where xx is the device code (truncated to two digits) specified in the DEVICE statement.

DRIVER LOGIC FLOW

Figure 9-3 shows the internal flow of an I/O operation starting with the Supervisor Call instruction executed by a running task and ending with the return through the scheduler to activate the highest priority ready task

```
                           ┌──────────────┐
                           │  USER TASK   │
                           └──────┬───────┘
                                  │←─── (SVC 1 INTERRUPT)
                           ┌──────┴───────┐
                           │    SVC 1     │
                           └──────┬───────┘
                                  │
                              ◇ DEVICE            ┌──────────┐      TASK PUT INTO I/O
                                BUSY  ──YES──────►│   BUSY   │      WAIT FOR THE
                                ?                 └──────────┘      DEVICE (UNLESS
                                │                                   UNCONDITIONAL
                               NO ◄──────────────────────┘         PROCEED OR TEST
                                                    (BRANCH)        I/O WAS SPECIFIED)
                                                                    UNTIL DEVICE IS
┌────────┐      ◇ FUNCTION   ┌──────────────┐                       MADE NON-BUSY BY
│ IOEXIT │◄─NO── CCPL ETC.   │    DRIVER    │                       IODONE
└────────┘        UP ?    ◄──│ INITIALIZATION│
                  │          │   ROUTINE     │
                 YES         └──────────────┘
                  └──────────────────────────── (SINT)
                           ┌──────────────┐
                           │/INTERRUPT/   │
                           │/SERVICE /    │
                           │/ROUTINE/     │
                           └──────┬───────┘
                                  │←─── ( LPSW DBxxxx )
                           ┌──────┴───────┐
                           │    DRIVER    │
                           │INITIALIZATION│
                           │   ROUTINE    │
                           └──────┬───────┘
                                  │←─── (BRANCH)
                           ┌──────┴───────┐
                           │    IOTWAT    │
                           └──────┬───────┘
                                  │←─── (BRANCH)
                           ┌──────┴───────┐
                           │    IOTERM    │
                           └──────┬───────┘
                                  │←─── (BRANCH)
                           ┌──────┴───────┐
                           │     TASK     │
                           │  SCHEDULER   │
                           └──────┬───────┘
                                  │←─── (LPSW)
                           ┌──────┴───────┐
                           │  USER TASK   │
                           └──────────────┘
```

SUSTAT X'7C00'
QUEUE INTERRUPT AND
PROTECT MODE DISABLED

ISSTAT X'3000'
MACHINE MALFUNCTION AND
FIXED PT DIVIDE FAULT
ENABLED

SUSTAT / ISSTAT   USER SUPPLIED
                   DRIVER

Figure 9-3   Internal Flow

INTERRUPT SERVICE

```
┌─────────────────┐
│ USER TASK        │
│ SYSTEM TASK      │
│ EXECUTIVE        │
└─────────────────┘
         │        ⟵ (IMMEIDATE INTERRUPT)
         ▼
┌─────────────────┐
│ DRIVER           │
│ INTERRUPT        │
│ SERVICE          │
│ RDUTINE          │
└─────────────────┘
         │        ⟵ (LPSW)
         ▼
┌─────────────────┐
│ USER TASK        │
│ SYSTEM TASK      │
│ EXECUTIVE        │
└─────────────────┘
```

I/O TERMINATION

```
┌─────────────────┐
│ USER TASK        │
│ SYSTEM TASK      │
│ EXECUTIVE        │
└─────────────────┘
         │        ⟵ (IMMEDIATE INTERRUPT)
         ▼
┌─────────────────┐        ┌──────────────────────────────┐
│ DRIVER           │        │ I/O COMPLETED                │
│ INTERRUPT        │        │ TERMINATION PHASE            │
│ SERVICE          │ ⟵──────│ ENTERED WITH ATL OF          │
│ ROUTINE          │        │ A(DCB)+1 ON SYSTEM           │
└─────────────────┘        │ QUEUE (LIOTRM) LPSW          │
         │                  │ WITH BIT 6 OF PSW SET.       │
         ▼                  │ SYSTEM QUEUE INTERRUPT       │
┌─────────────────┐        └──────────────────────────────┘
│ EXECUTIVE        │
└─────────────────┘
         │        ⟵ (BRANCH)
         ▼
┌─────────────────┐
│ IOTERM           │
└─────────────────┘
         │
         ▼
       ◇ DRIVER
       TERM         YES  ┌─────────────────┐
       ROUTINE ⟵─────────│ DRIVER          │
         ?                │ TERMINATION     │
         │NO              │ ROUTINE         │
         │                └─────────────────┘
         ▼
┌─────────────────┐
│ IODONE           │
│ ↓                │
└─────────────────┘
         │        ⟵ (BRANCH)
         ▼
┌─────────────────┐
│ IOTERM           │
│ (QUEUE EMPTY)    │
└─────────────────┘
         │        ⟵ (BRANCH)
         ▼
┌─────────────────┐
│ TASK             │
│ SCHEDULER        │
└─────────────────┘
         │        ⟵ (LPSW)
         ▼
┌─────────────────┐
│ USER TASK        │
└─────────────────┘
```

Figure 9-3. Internal Flow (Continued)

SUPERVISOR CALLS

All requests for I/O operations start with a Supervisor Call instruction, such as

        SVC   1,PBLK

in which the 1 in the R1 field indicates an I/O request and PBLK in the A(X2) field is the address of a parameter block. The parameter block must consist of a one byte function code and a one byte logical unit number. The meaning of bits within the function code is left for interpretation by the driver with the following exceptions:

1. Bit 0 must signify commands/data transfer

2. Either or Both of Bits 1 and 2 must be on for a data transfer

3. Bit 4 must signify wait or proceed

4. Bit 6 must be used for unconditional proceed

The logical unit number must be less than the SYSGEN parameter UNITS. The second halfword of the parameter block must be reserved as a location in which the system can return ending status and the device number on completion of the operation. The entire parameter block must be within the user desingated memory area. If Bit 0 of this function code specifies a data transfer, the length of data transferred (in bytes) is returned in the parameter block. See Figure 9-4 for the SVC 1 parameter block STRUC.

SVC1.	STRUC		
SVC1.FUN	DS	0	FUNCTION CODE/LOGICAL UNIT
SVC1.FC	DS	1	FUNCTION CODE
SVC1.LU	DS	1	LOGICAL UNIT
SVC1.STA	DS	0	STATUS
SVC1.DIS	DS	1	DEVICE INDEPENDENT STATUS
SVC1.DDS	DS	1	DEVICE DEPENDENT STATUS
SVC1.SAD	DS	2	STARTING ADDRESS
SVC1.EAD	DS	2	ENDING ADDRESS
SVC1.RAD	DS	4	RANDOM ADDRESS
SVC1.LXF	DS	2	LENGTH OF LAST TRANSFER
*	FUNCTION CODE		
S1FC.RDM	EQU	X'40'	READ
S1FC.WRM	EQU	X'20'	WRITE
S1FC.BIM	EQU	X'10'	BINARY/ASCII
S1FC.WTM	EQU	X'08'	WAIT/PROCEED
S1FC.RNM	EQU	X'04'	RANDOM/SEQUENTIAL
S1FC.UPM	EQU	X'02'	UNCONDITIONAL PROCEED
S1FC.IMM	EQU	X'01'	IMAGE/FORMATTED
S1FC CMM	EQU	X'80'	COMMAND FUNCTION
S1FC RWM	EQU	X'40'	REWIND
S1FC BRM	EQU	X'20'	BACKSPACE RECORD
S1FC FRM	EQU	X'10'	FORWARD SPACE RECORD
S1FC.WFM	EQU	X'08'	WRITE FILE MARK
S1FC.FFM	EQU	X'04'	FORWARD FILE
S1FC. BFM	EQU	X'02'	BACKWARD FILE
*	STATUS HALFWORD		
S1ST.ERM	EQU	X'8000'	ERROR STATUS
S1ST. IFM	EQU	X'4000'	ILLEGAL FUNCTION
S1ST.DUM	EQU	X'2000'	DEVICE UNAVAILABLE
S1ST.EMM	EQU	X'1000'	END OF MEDIUM
S1ST.EFM	EQU	X'0800'	END OF FILE
S1ST.UEM	EQU	X'0400'	UNRECOVERABLE ERROR
S1ST REM	EQU	X'0200'	RECOVERABLE ERROR
S1ST.LUM	EQU	X'0100'	ILLEGAL OR UNASSIGNED LU
	ENDS		

Figure 9-4. SVC 1 Parameter Block

EXECUTIVE ROUTINES

Several Executive routines are involved in starting an I/O operation Of prime interest to those writing drivers is the SVC 1 handler (SVC1). This routine performs the following functions:

1   Save the caller's registers.

2.   Picks up the logical unit number. The logical unit number must be less than UNITS If not, the routine returns illegal LU status, X'81', to the caller immediately

3   Gets the DCB address from the task's logical unit table.

4   If the function code specifies data transfer (bit 0 reset), it checks the function with the device attribute If any part of the function is unsupported, it returns illegal function status, X'C0', to the caller immediately. If the function code specifies a command and there are no supported functions (as described in the section entitled Device Control Block) specified, it returns to the caller with normal status.

5   Checks the device busy flag If set, it puts the caller in I/O wait for the device (unless unconditional proceed or test I/O is specified).

6   Sets the busy flag for all devices except the disc and a binary synchronous communication line The address of the DCB is put in the busy flag.

7   Branches to the driver. On entry to the driver, the following registers are set up

> R1 – Address of DCB
> R2 – Caller's TCB Pointer and I/O Wait Pointer
> R3 – Address of SVC 1 parameter block
> R4 – Function and logical unit
> R6 – Device number
> R7 – Zero (status register)
> R8 – Address of driver

DRIVER INITIALIZE ROUTINE (DIR)

The actual function of the driver initialize routine depends largely on the device. In general, the driver·

1   Performs further checking on the function code for validity and to determine the type of operation.

2.   Sets up general registers 8–15 as follows

R8-13	As needed for later user by the interrupt service routine of the driver
R14	Address of DCB
R15	Address of the interrupt service routine to be entered on the first interrupt

3.   Stores general registers 2-15 in the DCB at REG2 save area (DCB.RSAV).

4   Simulates an interrupt from the device to get to the interrupt service routine where actual I/O instructions can be used.

5   Returns to the Executive

The driver returns to the Executive at one of two locations. If all has gone well, and the interrupt service routine is activated, the driver returns to IOTWAT, a routine in the Executive that takes care of the I/O wait thread before exiting to the Task Scheduler If the driver determines that there is something wrong with the operation before going to the interrupt service routine, it returns through IOEXIT. On this return, the status register R7 must return with the appropriate status, usually illegal operation, X'C000' For either return, registers 1, 2, 3, 4, and 6 must be set up as they were on entry to the driver.

INTERRUPT SERVICE ROUTINE (ISR)

The interrupt service routine, like the initialize routine, is very device dependent It is activated in one of three ways·

1 By the initialize routine to start the device (SINT).

2 By subsequent interrupts from the device

3 By a Simulate Interrupt from the system (time-out).

Entry to the interrupt service routine is always through an interrupt, either simulated or real. The following sequence occurs on such an interrupt·

1 The current PSW is saved in the DCB old PSW save location (DCB.OPSW).

2. The new status, ISSTAT (X'3000', machine malfunction and fixed point divide fault interrupts enabled), is loaded from the new PSW status location of the DCB (DCB.NPSS).

3 The location counter points to the next location of the DCB (DCB.ENTR) which contains the sequence:

(DCB.ENTR)	STM	R8,IORSAV
	LM	R8,REG8xxxx
	BR	RF
(DCB.LEAV)	STM	R8,REG8xxxx
(DCB.NOPI)	LM	R8,IORSAV
	LPSW	DBxxxx

The DCB saves eight registers, starting with R8, at the location IORSAV. IORSAV is a reserved area of 16 bytes in the Executive. Then, eight registers are loaded from REG8xxxx, a location within the DCB register save area (DCB.RSAV+12), and the location in Register 15, as set up by the initialize routine is entered. What occurs here is device dependent. Usually, the interrupt service routine issues an output command to the device to get it started and sets up Register 15 for subsequent interrupts. Because only eight registers are saved, the ISR can only use R8-RF. References to R2-R7 can be made by accessing the save area corresponding to the register number in DCB.RSAV. The ISR must not execute any instructions other than list instructions, which cause software traps to be entered, (i.e., Multiply and Divide) as the traps run with external interrupts enabled. The interrupt service routine must exit by branching back to the DCB, either to the second store multiple instruction (DCB.LEAV), or to the second load multiple instruction (DCB.NOPI), depending on whether it has to save any registers. In the case where the interrupt service routine is started by the initialize routine, the load PSW instruction returns control, with external interrupts again enabled, to the initialize routine, at the instruction immediately following the simulate interrupt instruction.

When the interrupt service routine is activated as a result of a subsequent interrupt from the device, it checks device status, performs the necessary operation, and exits through the DCB On the final interrupt, the interrupt service routine executes an ATL instruction to put the address of the DCB+1 on the I/O termination queue, LIOTRM, and sets up the termination address, (DCB TERM), if required, before exiting through the DCB.

I/O TERMINATION

The system enters the I/O termination routine, IOTERM, as a result of an I/O termination queue interrupt. This interrupt is generated if there is something in the I/O termination queue, LIOTRM, and the processor executes a LPSW or EPSR instruction in which Bit 6 of the new status is set. The initialize routines, IOTWAT and IOEXIT, described previously, branch directly to IOTERM to avoid the overhead of an extra interrupt that would occur when the Task Scheduler executes a LPSW to schedule a task

The I/O termination queue interrupt is handled by the Executive, which saves the registers at the time of the interrupt and branches to IOTERM. IOTERM performs the following functions:

1. It removes the bottom entry from the list, LIOTRM.

2. If the list is empty it branches to SCHED, the Task Scheduler.

3. If the item removed is the address of DCB+1, it picks up the address of the driver termination routine from DCB.TERM.

4. It loads Registers 2 through 15 from the DCB. If the address of the driver termination routine is zero, it branches directly to IODONE, described below. However, if the address is non-zero, it puts the address of the termination routine in Register 8, and branches on Register 8.

When IOTERM branches to a driver termination routine, Register 1 contains the address of the DCB, Register 8 contains the address of the termination routine, and the remaining registers contain what was in the DCB locations from which they were loaded

Termination routines. like initialize and interrupt service routines are device dependent In many cases, they are not required at all. i.e.. the action taken by IODONE is sufficient to terminate the operation. The terminate routine can be used for code conversion, special error checking. or any other operations required after the transfer is complete that can be done with external interrupts enabled. Driver termination routines must exit by branching to IODONE

IODONE

IODONE is the common termination routine in the Executive. It expects Register 1 to contain the address of the DCB, Register 2 to contain the caller pointer, as set up by SVC 1, Register 3 to contain the SVC 1 parameter block address, Register 7 to contain the device independent status, Register 6 to contain the device dependent status, and Register 10 to contain the A(last byte transferred)+1. IODONE returns status and length of data transfer to the caller, takes all tasks in the I/O wait thread for that device out of I/O wait, and clears the I/O wait thread. If the device independent status (contents of Register 7) is non-zero, it returns the device dependent status, contents of Register 6, to the caller along with the device independent status IODONE exits through IOTERM, to remove all items from the queue, before returning to the Scheduler Any tasks that were in I/O Wait for the device are now rescheduled by priority

I/O WAIT

Tasks which require I/O on a device that is busy (and the function code specified neither unconditional proceed nor test I/O) are put on the I/O wait thread for that device (see Figure 9-5). The thread starts in the right byte of REG2xxxx in the DCB Its contents are the TCB pointer of the next task in the thread The thread continues through the TCBs of all tasks in the thread. The end of the thread is X'00'.

Register 2 is initialized by SVC 1 with the caller's TCB pointer in the left byte and X'00' in the right byte. It must be saved in REG2xxxx in the DCB by the driver in the initialization phase. The thread is built by SVC 1 as other tasks require that device The user written driver must maintain the integrity of REG2xxxx. If this location is overwritten the thread is destroyed.
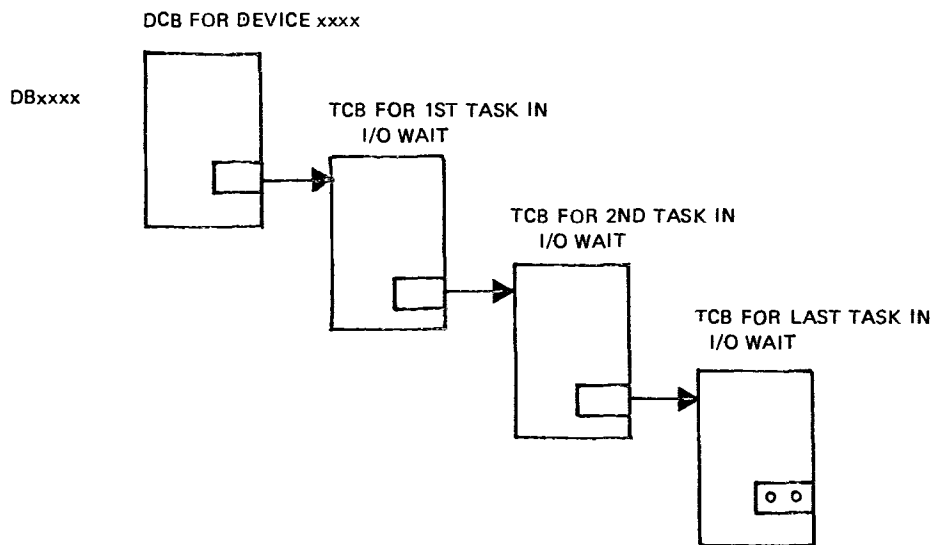
Figure 9-5. I/O Wait Thread

CANCEL I/O

When a task is cancelled, either by a console CANCEL or an SVC 6 task cancel, or goes to End of Task (SVC 3), and I/O is on-going for that task, the EOT routine gets the DCB driver abort routine address (DCB ALOC). It stores it in the Register 15 save area in the DCB (DCB.ISR), and executes a simulate interrupt instruction (SINT). The abort routine is entered in ISSTAT state, (X'3000') with the registers initialized from REG8xxxx (save area in DCB). The logic of the driver abort routine depends largely on the device, but it generally issues a STOP I/O command to the device, sets an unrecoverable error, schedules the I/O termination routine (ATL to LIOTRM), and exits via the DCB (DCB.LEAV).

HALT I/O

When a task requests Halt I/O to a device that supports it, and I/O is on-going, the driver's abort routine is entered, as if the task was cancelled, except that the Halt I/O flag (DFLG.HIM) is set in DCB.FLGS. If the abort routine finds this flag set, it returns Halt I/O error status (X'8281') instead of an unrecoverable error status

SAMPLE DRIVER

The following driver is similar to the standard OS/16 MT2 Card Reader Driver It is presented as a coding sample for user written drivers The comments on the listing reflect the functions described in the previous narrative

```
*               THIS PROGRAM IS A CARD READER DRIVER.
*               IT READS UP TO 80 ASCII CHARACTERS
*               OR 160 IMAGE CHARACTERS
*               IT CONVERTS THE HOLLERITH DATA
*               PRODUCED BY AN IBM029 OR
*               IBM 026 KEYPUNCH.
*
*               EXTRN/ENTRY DECLARATIONS
*               EXTRN  LIOTRM,IOTWAT,IODONE,IOEXIT
*               ENTRY  CRDDVR,CRDTRM
*
*               REGISTER ALLOCATION
*               COPY   REGS
*               COPY   DCB
*               COPY   SVC1.
HOLBUF          EQU    DCB.SDN
*               ENTERED HERE ON SVC IF NOT BUSY
*               INITIALIZATION PHASE
CRDDVR          LBR    RD,R6            MOVE DEVICE NUMBER TO RD
                SSR    RD,R0            SENSE STATUS
                THI    R0,X'20'         IS HOPPER EMPTY?
                BZS    CRDVRB           NO READ A CARD
                LHI    R7,X'A000'       HOOPER EMPTY = D. U.
                B      IOEXIT
*
CRDVRB          LHI    RA,HOLBUF(R1)    SET STARTING ADRS OF BUFF
                LIS    RB,2             SET BXH INCREMENT VALUE
                LHR    RE,R1            ADRS OF DCB INTO R14
                LHI    R8,CRTERM        SET UP TERMINATION ROUTINE ADDRESS
                STH    R8,DCB.TERM(RE)
                LHI    RC,159(RA)       SET END ADRS OF CARD BUFF
                LHI    RF,CRISR         SET INTERRUPT SERVICE ROUTINE ADDR
                STM    R2,DCB.RSAV(R1)  SAVE DRIVER REGISTERS
                SINT   0(RD)            START I/O OPERATION
                B      IOTWAT           EXIT INITIALIZATION PHASE
*               ON I/O INTERRUPT CONTROL TRANSFERS HERE, INTS DISABLED.
*               INTERRUPTS SERVICE ROUTINE PHASE
CRISR           LHI    RF,CRISR1        SET ISR POINTER
                OC     RD,CFEED         FEED A CARD
                B      DCB.LEAV(RE)     RETURN AND WAIT FOR FIRST INT
```

```
CRISR1     SSR    RD,R8                   CHECK STATUS
           BFC    1,NOTDU                 BRANCH IF DEVICE AVAILABLE
CRDTRM     EQU    *                       (THE ABORT ROUTINE IS HERE)
ERR        LHI    RB,X'8400'              UNREC ERROR STATUS
           STH    RB,DCB.RSAV+10(RE)      SAVE IT IN R7 SAVE AREA
           XHR    R8,R8                   IODONE RETURN
           STH    R8,DCB.TERM(RE)         ZERO TERMINATION ROUTINE ADDR
BACK       OC     RD,DISABL               DISABLE INTERRUPTS
           LHI    RF,DCB.NOPI(RE)         SET NOP ISR
           AIS    RE,1                    MAKE DCB ADRS ODD
           ATL    RE,LIOTRM
           B      DCB.LEAV-1(RE)          RETURN
NOTDU      BNOS   CHKSTS                  EXAMINE SET?
           THI    R8,X'D0'                ANY BAD STATUS?
           BNZ    ERR                     YES
CHKSTS     SSR    RD,R8                   GET STATUS SOON
           NHI    R8,X'20'                YES. HOPPER EMPTY SET?
           BZ     ERR                     NO  UNREC ERR
BSY        SSR    RD,R8                   GET STATUS AGAIN
BSY2       BTC    8,DCB.NOPI(RE)          BSY=1?
           RH     RD,0(RA)                READ 2 BYTES
           BXH    RA,BACK                 LOOP FOR ENTIRE CARD
SAVE       STH    RA,DCB.RSAV+16(RE)      SAVE CURRENT BUFFER POINTER ONLY
           B      DCB NOPI(RE)            RETURN
*
CFEED      DB     X'60'                   FEED A CARD
DISABL     DB     X'80'                   DISABLE INTS
*                 TERMINATION ROUTINE -- CONVERT CARD
CRTERM     LH     R8,SVC1 SAD(R3)         GET USER BUFFER START ADDR
           LIS    R9,1                    SET BXLE LOOP
           LH     RA,SVC1.EAD(R3)         GET USER BUFFER END ADDR
           LHI    RC,HOLBUF(R1)           GET DRIVER BUFFER ADDR
           THI    R4,X'0100'              ASCII OR IMAGE
           BZS    CRASCI
*                 HERE MOVE DATA DIRECTLY FROM DRIVER BUFFER TO USER BUFFER
           B      LENGTH
*
CRASCI     EQU    *
*                 HERE CONVERT EACH HOLLERITH HALFWORD
*                 TO AN ASCII BYTE AND STORE IN USER BUFFER
LENGTH     LHR    RA,R8                   RA=A(LAST BYTE TRANSFERRED)+1
                                          USED BY IODONE TO CALCULATE
                                          LENGTH OF XFER
           B      IODONE                  END TERMINATION PHASE
```

This sample driver can be assembled with the following sequence of commands, assuming the source is on cards

```
LD  CAL16
TA  .BG
AL  CRDDVR.OBJ,IN          ;*ALLOCATE OBJECT FILE
AL  SCRAT,IN,80            ;*ALLOCATE SCRATCH FILE
AL  CROSS,IN,256           ;*ALLOCATE CROSS FILE
AS  1,CR                   ;*ASSIGN INPUT DEVICE
AS  2,CRDDVR OBJ           ;*ASSIGN OUTPUT FILE
AS  3,PR.                  ;*ASSIGN LIST DEVICE
AS  4,SCRAT                ;*ASSIGN SCRATCH FILE
AS  5,CROSS                ;*ASSIGN CROSS FILE
AS  7,PCB16.CAL            ;*ASSIGN COPY FILE
START                      ;*START CAL16
```

SUMMARY OF REGISTER CONVENTIONS

Registers On Entry To Driver

The driver is entered with the following registers set up.

Driver Initialization Routine

R1 = A(DCB)
R2 = caller TCB pointer, left byte
X'00' right byte
R3 = A(SVCI parameter block)
R4 = Function Code, left byte
Logical Unit, right byte
R6 = Device Number
R7 = 0 Status
R8 = A(Driver initialization routine)

Driver Interrupt Service Routine

R8-R15 initialized with REG8xxxx (save area in DCB)

Driver Termination Routine

R2-R15 initialized with REG2xxxx (save area in DCB)

Registers On Entry To Executive

The driver must insure the following registers are set up before branching to the Executive entry points.

IOTWAT

R1 = A(DCB)
R2 = caller TCB pointer, left byte
X'00', right byte
R3 = A(SVCI parameter block)
R4 = Function code, left byte
Logical unit, right byte
R6 = Device number
R7 = 0 status

IOEXIT

Same as IOTWAT except for
R6 = device dependent status
R7 = device independent status

IODONE

R1 = A(DCB)
R2 = caller TCB pointer, left byte
I/O wait pointer, right byte
R3 = A(SVC 1 parameter block)
R6 = device dependent status
R7 = device independent status
R10 = A(last byte transferred)+1
used only on data transfer requests

RTOS DRIVER COMPATIBILITY WITH OS/16 MT2

The following is a list of items for consideration when converting an existing RTOS driver for user under OS/16 MT2

1. The names of the Driver Initialize Routine and the Driver Abort Routine must be of the form xxxDVR and xxxTRM.

2 The abort routine must be modified to return an unrecoverable error for recovery from a power failure.

3 The abort routine may be modified to support Halt I/O, if desired

4 Branching to DCB LEAV always saves R8-RF. If fewer registers are to be saved, the ISR must save them itself and exit to DCB NOPI.

5 When IODONE is entered, RA must contain the address of the last byte transferred plus one, or the SVC1 LXF field will not be set up properly

APPENDIX 1
SVC SUMMARY

General form of an SVC instruction is:

   SVC N, P

N — Particular SVC number
P — Effective address of parameter block or parameter

SVC parameter blocks must be aligned on a halfword boundary.

Function	SVC
I/O Requests	SVC 1
Task Control	
Pause self	SVC 2 Code 1
End self	SVC 3 or SVC 6 End Task
Load another foreground task	SVC 6 Load
Start another foreground task	SVC 6 Start
Cancel another foreground task	SVC 6 End task
Send a message to another task	SVC 6 Send Message
Add parameter to another foreground task's Task Queue	SVC 6 Queue parameter
Add parameter to own Task Queue	SVC 6 Queue parameter
Change another foreground task's priority	SVC 6 Change priority
Change own priority	SVC 6 Change priority
Connect Trap Generating Device to task	SVC 6 Connect
Enable Trap Generating Device for interrupts	SVC 6 Thaw
Disable Trap Generating Device interrupts	SVC 6 Freeze
Cause a Trap Generating Device interrupt	SVC 6 SINT
Disconnect Trap Generating Device from task	SVC 6 Unconnect
Delete task from memory	SVC 6 Task Non Resident and End task
Make task memory resident	SVC 6 Task Resident
Load and Start task	SVC 6 Load and Start
Find Task Status	SVC 6 no function
Find Task priority (current)	SVC 6 no function
Find Task ID of self	SVC 2 Code 19
Find Task options of self	SVC 2 Code 19
Find Current TSW for self	SVC 2 Code 19
Find UDL	SVC 2 Code 5
Enable traps	SVC 9
Enable Task Queue entries	SVC 9
Disable Task Queue entries	SVC 9
Wait for trap	SVC 9
Memory Management	
Get storage from allocated memory	SVC 2 Code 2
Return gotten storage to allocated memory	SVC 2 Code 3
Find memory limits for task	SVC 2 Code 5
Fetch overlay into memory	SVC 5
Utility Functions	
Output message to system console	SVC 2 Code 7
Convert from ASCII Hexadecimal to Binary	SVC 2 Code 15
Convert from ASCII Decimal to Binary	SVC 2 Code 15
Convert from Binary to ASCII Hexadecimal	SVC 2 Code 6
Convert from Binary to ASCII Decimal	SVC 2 Code 6
Convert ASCII File Descriptor to SVC 7 format	SVC 2 Code 16
Search for occurrence of ASCII String in a table	SVC 2 Code 17
Move ASCII characters up to some ending character	SVC 2 Code 18
Change PSW Condition Code	SVC 2 Code 4
Disable or Enable Arithmetic Faults	SVC 2 Code 4

Clock Services
  Fetch Date (ASCII)   SVC 2 Code 9
  Fetch Time (ASCII or Binary)   SVC 2 Code 8
  Wait for specified time of day   SVC 2 Code 23
  Wait for specified number of milliseconds   SVC 2 Code 23
  Take a trap at specified time of day   SVC 2 Code 23
  Take a trap in specified number of milliseconds   SVC 2 Code 23
  Cancel a time trap   SVC 2 Code 23

File Management
  Allocate a direct access file on a device   SVC 7 Allocate
  Assign a device/file to a Logical Unit   SVC 7 Assign
  Deassign a Logical Unit   SVC 7 Close
  Delete a direct access file from a volume   SVC 7 Delete
  Change the protection keys of a file   SVC 7 Assign, Reprotect, and Close
  Change the name of a direct access file   SVC 7 Assign, Rename, and Close
  Change the name of a device (E-task only)   SVC 7 Assign, Rename, and Close
  Delete an unconditionally protected file   SVC 7 Delete
    (Keys of FF) (E-task only)
  Flush output buffers of buffered file or device   SVC 7 Checkpoint
  Find the maximum record length of device/file   SVC 7 Fetch Attributes
    assigned to a Logical Unit
  Find File Descriptor of device/file assigned to   SVC 7 Fetch Attributes
    a Logical Unit
  Append Data to Indexed File   SVC 7 Assign for EWO or SWO
  Find Number of Logical Units available to task   SVC 2 Code 19

SVC 1 -- INPUT/OUTPUT REQUEST

CAL CODE   FORMAT

PARBLK

ALIGN	2
DB	fc,lu
DC	H'0'
DAC	START
DAC	END
DC	Y'random'
DAS	1
DC	Y'EXTOPT' (ITAM ONLY)

0(0) FC	1(1) LU
2(2) STATUS	
4(4) A(START)	
6(6) A(END)	
8(8) RANDOM	
12(C) LENGTH OF XFR	
14(E) (4 bytes) EXT.OPT. (ITAM)	

CONDITION CODE

0   NORMAL COMPLETION
F (ALL BITS)   UNCONDITIONAL PROCEED REQUEST FOR BUSY FILE/DEVICE.

```
┌─────────────────────────────┐
│ SVC 2 CODE 1 — PAUSE        │
└─────────────────────────────┘
```

CAL CODE   FORMAT

              ALIGN 2
PARBLK   DB   0,1

0(0)	1(1)
00	1 (01)

NO OPTIONS SUPPORTED
CONDITION CODE UNCHANGED.

```
┌─────────────────────────────────┐
│ SVC 2 CODE 2 — GET STORAGE      │
└─────────────────────────────────┘
```

CAL CODE   FORMAT

              ALIGN 2
PARBLK   DB   OPTION,2
              DC   H'REG'
              DAC   SIZE

0(0)	1(1)
OPTION	2(02)
2(2) REG	
4(4) SIZE	

OPTIONS   MEANING

X'00' —   GET NUMBER OF BYTES SPECIFIED BY SIZE
X'80' —   GET ALL ALLOCATED MEMORY — NUMBER OF BYTES GOTTEN
                  OR RETURNED IN SIZE

REG — SPECIFIES GENERAL REGISTER USED TO RETURN STARTING ADDRESS OF
      GOTTEN MEMORY.

CONDITION CODE

      0 —   NORMAL COMPLETION
      4(V-BIT) — REQUESTED SIZE TOO LARGE OR NEGATIVE. REG SET TO ZERO.

```
┌─────────────────────────────────────┐
│  SVC 2 CODE 3 — RELEASE STORAGE      │
└─────────────────────────────────────┘
```

CAL CODE   FORMAT

ALIGN 2

PARBLK   DB   0,3

DAC   SIZE

0(0)	1(1)
00	3(03)
2(2) SIZE	

NO OPTIONS

CONDITION CODE   MEANING

0   NORMAL COMPLETION
4 (V-BIT)   SIZE TOO LARGE OR NEGATIVE

```
┌─────────────────────────────────┐
│  SVC 2 CODE 4 — SET STATUS       │
└─────────────────────────────────┘
```

CAL CODE   FORMAT

ALIGN 2

PARBLK   DB   OPTION,4

DB   AF,CC

0(0) OPTION	1(1) 04(04)
2(2) AF	3(3) CC

OPTION (HEX)   MEANING

00   MODIFY STATUS AND CONDITION CODE
80   MODIFY CONDITION CODE ONLY

AF BYTE (HEX)   MEANING

00   DISABLE ARITHMETIC FAULTS
10   ENABLE FIXED POINT DIVIDE FAULT ONLY
04   ENABLE FLOATING POINT ARITHMETIC FAULT
14   ENABLE FIXED POINT DIVIDE AND
                            FLOATING POINT ARITHMETIC FAULTS

CC BYTE (HEX)   MEANING

0x   x IS TO REPLACE CONDITION CODE

CONDITION CODE

VALUE SPECIFIED IN CC BYTE.

```
┌─────────────────────────────────────┐
│ SVC 2 CODE 5 – FETCH POINTER         │
└─────────────────────────────────────┘
```

CAL CODE   FORMAT

ALIGN   2
PARBLK   DB   0,5
         DC   H'REG'

```
┌──────────────────┬──────────────────┐
│ 0(0)             │ 1(1)             │
│            00    │            5(05) │
├──────────────────┴──────────────────┤
│ 2(2)                                 │
│            REG                       │
└──────────────────────────────────────┘
```

NO OPTIONS

CONDITION CODE
    UNCHANGED

```
┌─────────────────────────────────────────┐
│ SVC 2 CODE 6 – UNPACK BINARY NUMBER       │
└─────────────────────────────────────────┘
```

CAL CODE   FORMAT

ALIGN   2
PARBLK   DB   OPTION+N,6
         DAC   DEST

```
┌──────────────────┬──────────────────┐
│ 0(0              │ 1(1)             │
│       OPTION+N   │            6(06) │
├──────────────────┴──────────────────┤
│ 2(2)                                 │
│            A(DEST)                    │
└──────────────────────────────────────┘
```

OPTION (HEX)   MEANING

    00   CONVERT TO HEXADECIMAL
    40   CONVERT TO HEXADECIMAL, SUPPRESS LEADING ZEROS
    80   CONVERT TO DECIMAL
    C0   CONVERT TO DECIMAL, SUPPRESS LEADING ZEROS

N – LENGTH OF DESTINATION IN BYTES; IF ZERO, 4 IS ASSUMED.

NUMBER TO BE CONVERTED IS PASSED IN GENERAL REGISTER ZERO.

CONDITION CODE
    UNCHANGED

```
SVC 2 CODE 7 -- LOG MESSAGE
```

CAL CODE			
	ALIGN	2	
PARBLK	DB	OPTION,7	(a)
	DC	H'LENGTH'	DIRECT TEXT
	DC	C'TEXT'	

FORMAT

0(0)	1(1)	
OPTION		7(07)
2(2) LENGTH		
4(4) TEXT		

CAL CODE			
	ALIGN	2	
PARBLK	DB	OPTION,7	(b)
	DC	H'LENGTH'	INDIRECT TEXT
	DAC	TEXT	

FORMAT

0(0)	1(1)	
OPTION		7(07)
2(2) LENGTH		
4(4) A(TEXT)		

OPTIONS (HEX)	MEANING
00	DIRECT TEXT, FORMATTED
40	INDIRECT TEXT, FORMATTED
80	DIRECT TEXT, IMAGE
C0	INDIRECT TEXT, IMAGE

CONDITION CODE

UNCHANGED

```
SVC 2 CODE 8 -- FETCH TIME
```

CAL CODE		
	ALIGN	2
PARBLK	DB	OPTION,8
	DAC	DEST

FORMAT

0(0)	1(1)	
OPTION		8(08)
2(2) A(DEST)		

OPTIONS (HEX)	MEANING
00	ASCII FORMAT (hh:mm:ss)
80	BINARY FORMAT (4 BYTES--SECONDS SINCE MIDNIGHT)

CONDITION CODE

UNCHANGED

```
┌─────────────────────────────┐
│ SVC 2 CODE 9 – FETCH DATE   │
└─────────────────────────────┘
```

	CAL CODE	
	ALIGN	2
PARBLK	DB	0,9
	DAC	DEST

FORMAT

0(0)	1(1)
00	9(09)
2(2)	
A(DEST)	

NO OPTIONS – DATE RETURNED IN ASCII mm/dd/yy (or dd/mm/yy according to SYSGEN)

CONDITION CODE

    UNCHANGED

```
┌──────────────────────────────────────┐
│ SVC 2 CODE 15 – PACK NUMERIC DATA     │
└──────────────────────────────────────┘
```

	CAL CODE	
	ALIGN	2
PARBLK	DB	OPTION,15
	DC	H'REG'

FORMAT

0(0)	1(1)
OPTION	15(0F)
2(2)	
REG	

OPTIONS (HEX)	MEANING
00	HEXADECIMAL
40	HEXADECIMAL, SKIP LEADING BLANKS
80	DECIMAL
C0	DECIMAL, SKIP LEADING BLANKS

BINARY RESULT RETURNED IN GENERAL REGISTER ZERO.
REG – SPECIFIES GENERAL REGISTER CONTAINING ADDRESS OF ASCII STRING TO
               BE CONVERTED. RETURNED POINTING TO FIRST BYTE NOT CONVERTED.

CONDITION CODE	MEANING
0	NORMAL COMPLETION
1 (L-BIT)	NO CHARACTERS PROCESSED, GENERAL REGISTER ZERO SET TO ZERO.
4 (V-BIT)	OVERFLOW, MORE THAN 4 HEXADECIMAL DIGITS OR DECIMAL GREATER THAN $2^{16} - 1$ (65,535). RESULT IS VALUE MODULO 10000 (HEX) OR 2^{16} (DECIMAL)

SVC 2 CODE 16 – PACK FILE DESCRIPTOR

CAL CODE

```
           ALIGN   2
PARBLK     DB      OPTION,16
           DC      H'REG'
           DAC     DEST
```

FORMAT

0(0) OPTION	1(1) 16(10)
2(2) REG	
4(4) A(DEST)	

OPTIONS (HEX)	MEANING
00	DEFAULT SYSTEM VOLUME
40	DEFAULT SYSTEM VOLUME, SKIP LEADING BLANKS
80	NO DEFAULT VOLUME
C0	NO DEFAULT VOLUME, SKIP LEADING BLANKS

RESULT FORMAT

0(0) VOLUME NAME	
4(4) FILENAME	
12(C) EXTENSION	15(F) BLANK (X'20')

REG = SPECIFIES GENERAL REGISTER POINTING TO ASCII STRING TO BE CONVERTED. RETURNED POINTING TO FIRST BYTE NOT CONVERTED

CONDITION CODE	MEANING
0	NORMAL COMPLETION
1 (L-BIT)	NO VOLUME NAME PRESENT IN INPUT
4 (V-BIT)	SYNTAX ERROR
8 (C-BIT)	NO EXTENSION PRESENT IN INPUT
9 (L & C BITS)	NO VOLUME, NO EXTENSION IN INPUT

```
┌─────────────────────────────────────────────┐
│ SVC 2 CODE 17 – MNEMONIC TABLE SCAN           │
└─────────────────────────────────────────────┘
```

CAL CODE   FORMAT

PARBLK

ALIGN	2
DB	0,17
DB	REG1,REG2
DAC	TABLE

0(0)	1(1)
00	17(11)
2(2) REG1	3(3) REG2
4(4) A(TABLE)	

NO OPTIONS

REG1 – SPECIFIES GENERAL REGISTER POINTING TO ASCII STRING TO BE SOURCE SCANNED.
RETURNED POINTING TO FIRST BYTE NOT IN MNEMONIC.

REG2 – SPECIFIES GENERAL REGISTER USED TO RETURN INDEX OF MATCHED MNEMONIC
FROM 0 TO NUMBER OF MNEMONICS – 1. –1 INDICATES NO MATCH.

TABLE – A TABLE OF ASCII MNEMONICS SEPARATED FROM EACH OTHER BY A BYTE OF
ZERO. REQUIRED CHARACTERS FOR EACH MNEMONIC ARE INDICATED BY SETTING
HIGH ORDER BIT. TABLE TERMINATED WITH TWO CONSECUTIVE BYTES OF ZERO

CONDITION CODE	MEANING
0	NORMAL COMPLETION
4 (V-BIT)	NO MATCH FOUND, REG2 SET TO –1

```
┌─────────────────────────────────────────────┐
│ SVC 2 CODE 18 – MOVE ASCII CHARACTERS         │
└─────────────────────────────────────────────┘
```

CAL CODE   FORMAT

PARBLK

ALIGN	2
DB	OPTION + N,18
DB	REG1,REG2
DAC	ECSTRING

0(0) OPTION+N	1(1) 18(12)
2(2) REG1	3(3) REG2
4(4) A(ECSTRING)	

OPTIONS (HEX)	MEANING
00	NO ENDING CHARACTERS
80	USE ENDING CHARACTER STRING TO TERMINATE

N – MAXIMUM NUMBER OF CHARACTERS TO MOVE

REG1 – SPECIFIES GENERAL REGISTER POINTING TO ASCII STRING TO BE MOVED.
RETURNED POINTING TO FIRST BYTE NOT MOVED.

REG2 – SPECIFIES GENERAL REGISTER POINTING TO DESTINATION. RETURNED POINTING
TO BYTE FOLLOWING LAST BYTE MOVED INTO DESTINATION.

ECSTRING – STRING OF TERMINATION CHARACTERS OF FORM:

```
0        1                    m+1
┌────────┬──────────────────────┐
│   M    │  ENDING CHARACTERS    │
└────────┴──────────────────────┘
```
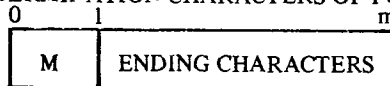
M – NUMBER OF ENDING CHARACTERS

CONDITION CODE	MEANING
0	NORMAL COMPLETION
4 (V-BIT)	NO ENDING CHARACTER FOUND (OPTION X'80')

```
                    SVC 2 CODE 19 — PEEK
```

CAL CODE

	ALIGN	2
PARBLK	DB	0,19
	DS	24

FORMAT

0(0)	1(1)
00	19(13)
2(2)	3(3)
NLU	MPRI
4(4)	
OS ID	
12(C)	
TASK ID	
20(14)	
CTSW	
22(16)	
TASK OPTIONS	
24(18)	
TASK STATUS	

NO OPTIONS

FIELDS RETURNED·

NLU	—	NUMBER OF LUs AVAILABLE TO TASK
MPRI	—	MAXIMUM PRIORITY OF TASK
OS/ID	—	ASCII ID OF OS—OS16MT2r WHERE r INDICATES REVISION LEVEL
TASK ID	—	EIGHT-CHARACTER ASCII TASK ID
CTSW	—	STATUS PORTION (FIRST 16 BITS) OF TASK'S CURRENT
		TASK STATUS WORD
TASK		
OPTIONS	—	OPTIONS HALFWORD FROM TASK CONTROL BLOCK
TASK		
STATUS	—	STATUS HALFWORD FROM TASK CONTROL BLOCK

CONDITION CODE
    UNCHANGED

```
┌─────────────────────────────────────┐
│ SVC 2 CODE 23 – TIMER MANAGEMENT     │
└─────────────────────────────────────┘
```

CAL CODE   FORMAT

```
            ALIGN  2
PARBLK      DB     OPTION,23
            DC     H'REG'
            DC     Y'TIME'
```

0(0) OPTION	1(1) 23(17)
2(2) REG	
4(4) TIME	

OPTIONS (HEX)	MEANING
80	WAIT FOR INTERVAL COMPLETION
00	PROCEED AND ADD PARAMETER TO TASK QUEUE ON INTERVAL COMPLETION
10	CANCEL TIME INTERVAL

REG – SPECIFIES GENERAL REGISTER CONTAIN PARAMETER TO BE ADDED TO TASK QUEUE ON COMPLETION OF INTERVAL (OPTION X'00')

TIME –

BITS	MEANING
0-3	0000 – SECONDS SINCE MIDNIGHT (TIME OF DAY INTERVAL) 0001 – MILLISECONDS FROM NOW (ELAPSED TIME INTERVAL) OTHER – ILLEGAL
4-31	NUMBER OF CLOCK TICKS IN INTERVAL TICK = 1 SECOND FOR TIME OF DAY INTERVAL 1 MILLISECOND FOR ELAPSED TIME INTERVAL (OPTIONS X'00' AND X'80')

CONDITION CODE

UNCHANGED	FOR OPTIONS X'00' AND X'80'
0	FOR OPTION X'10' IF SUCCESSFUL
4(V-BIT)	FOR OPTION X'10' IF NO INTERVAL FOUND

```
┌─────────────────────────────┐
│ SVC 3 – END OF TASK (EOT)   │
└─────────────────────────────┘
```

SVC   3,N

NO PARAMETER BLOCK, SVC EFFECTIVE ADDRESS FIELD (N) STORED IN TASK'S RETURN CODE.

```
┌─────────────────────────┐
│ SVC 5 – FETCH OVERLAY   │
└─────────────────────────┘
```

CAL CODE   FORMAT

```
            ALIGN   2
PARBLK      DC      C'OVLYNM'
            DS      2
            DS      1
            DB      OPTION
            DC      H'LU'
```

```
┌─────────────────────────────┐
│ 0(0)                        │
│          OVERLAY            │
│           NAME              │
│         (6 BYTES)           │
├─────────────────────────────┤
│ 6(6)                        │
│          RESERVED           │
├──────────────┬──────────────┤
│ 8(8)         │ 9(9)         │
│   STATUS     │   OPTION     │
├──────────────┴──────────────┤
│ 10(A)                       │
│            LU               │
└─────────────────────────────┘
```

OPTIONS (HEX)	MEANING
01	LOAD FROM SPECIFIED LU WITHOUT POSITIONING
04	LOAD FROM SPECIFIED LU AFTER REWIND

STATUS (HEX)	MEANING
00	NORMAL COMPLETION
10	LOAD FAILED
20	MISMATCH ON OVERLAY NAME
40	OVERLAY WOULD NOT FIT IN ALLOCATED MEMORY

CONDITION CODE

UNCHANGED

SVC 6 – INTERTASK COORDINATION

CAL CODE

```
         ALIGN   2
PARBLK   DC      C'TASK ID'
         DC      Y'FUNCTION'
         DS      4
         DB      LU
         DB      PRI
         DS      2
         DC      A(START)
         DC      Y'TIME'
         DC      C'DEVM'
         DC      X'PARM'
         DC      A(MESSAGE)
         DS      4
```

FORMAT

0(0) TASK ID	
8(8) FUNCTION CODE	
12(C) TASK STATUS	
14(E) ERROR STATUS	
16(10) LOAD LU	17(11) PRIORITY
18(12) RPRI	19(13) RESERVED
20(14) START ADDRESS	
22(16) TT DELAY TIME	
26(1A) DEVICE MNEMONIC	
30(1E) PARAMETER	
32(20) MESSAGE BUFFER ADDRESS	
34(22) RESERVED (4 BYTES)	

SVC1   Function Codes

Function Code -- Data Transfer Request

Bit	Alignment	Meaning
0	0.......	This bit must be ZERO to indicate a data transfer request.
1-2	.xx. . .	Read-Write bits. The meaning of these two bits is modified by Bits 3-7 to control the transfer. Basically the values are: 10 — Read request 01 — Write request 11 — Test and Set request (see Chapter 8) 00 — Wait Only or Test I/O Complete
3	. .x . . .	ASCII/BINARY bit. This bit indicates the type of formatting requested if Bit 7 is reset. 0 — indicates ASCII formatting 1 — indicates Binary formatting If Bit 7 is reset, this bit must be ZERO.
4	. . . . x. . .	PROCEED/WAIT bit. This bit indicates the action to be taken after the I/O has been initiated 0 — Proceed Indicates that control is to be returned to the task after initiation of I/O. 1 — Wait. Indicates that the task is to be put into I/O Wait until the data transfer is complete
5	. . . . .x. .	SEQUENTIAL/RANDOM bit. 0 — Sequential. Indicates the next logical record is to be accessed. 1 — Random. Indicates the logical record specified by the RANDOM field is to be accessed.
6	. . .x.	UNCONDITIONAL PROCEED bit. 0 — indicates the task is to be put into I/O wait until the requested device/file is free. At that time the request is processed 1 — indicates that the request is to be rejected with a condition code of X'F' if the requested device/file is not free.
7	x	FORMATTED/IMAGE bit 0 — indicates that the request is to be formatted according to the device/file and the setting of Bit 3. 1 — indicates that no formatting is to be performed (Image mode).

Function Code — Command Function Request

Bit	Alignment	Meaning
0	1.......	This bit must be set to indicate a command function request. This bit alone set requests Halt I/O.
1	.x.. . .	Rewind
2	. .x. . . .	Backspace Record
3	. . .x	Forward Space Record
4	. x. . .	Write File Mark
5	x	Forward Space File
6	. . . . .x.	Backspace File
7	x	Reserved for driver dependent functions

Error Status Definition (SVC 1)

Bit	Meaning if set to 1	Binary	Hexadecimal
0	Always 1 for error status		
1	Illegal Function	1100 0000	X'C0'
2	Device Unavailable	1010 0000	X'A0'
3	End of Medium	1001 0000	X'90'
4	End of File	1000 1000	X'88'
5	Unrecoverable Error	1000 0100	X'84'
6	Parity or Recoverable Error	1000 0010	X'82'
7	Illegal or Unassigned LU	1000 0001	X'81'

SVC 6 Parameter Block Fields

Byte	Name	Meaning
0-7	Task ID	Name of task; not required if call is self-directed
8-11	Function Code	Specifies desired functions
12-13	Task Status	The wait status halfword of the specified task is returned by the system in this field.
14-15	Error Status	Set to ZERO for normal termination or to error code if error detected
16	Load LU	Specifies LU from which to load the task. Used only for Load function.
17	Priority	Specifies priority for change priority function. May not be 255 (E-tasks). Must be in range 10-249 (U-tasks)
18	RPRI	Set by system to actual priority of specified task.
19		(reserved)
20-21	Start Address	Used only for START functions. specifies address at which to start specified task. If ZERO, signifies normal transfer address, as specified at TET/16 time.
22 (Bits 0-3)	TT	Time Type indicates type of time delay. Used only for Delay-Start. Legal codes are: 0000 = seconds since midnight 0001 = milliseconds from now.
22-25 (Bits 54-31)	Delay Time	Specifies number of seconds or milliseconds, as defined by TT field, to delay the start operation. Used only for Delay-Start.
26-29	Device Mnemonic	Specifies device affected by Connect, Thaw, SINT, Freeze and Unconnect functions.
30-31	Parameter	Specifies parameter to be queued for Queue parameter function; specifies device parameter for Connect functions.
32-33	A(message)	Specifies address of message buffer for Send Message function.
34-37		(Reserved)

BITS	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	3	3
	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
NAME	D		E			L		H			M	Q	P			O	T	I	F	U						N				A		

BIT(S)	NAME	HEX MASK	MEANING	
0-1	D	8000 0000	Direction	00,01 = illegal codes
		C000 0000		10 = other task
				11 = self
2-3	E	0000 0000	End Task·	00 = no function requested
		1000 0000		01 = cancel
		2000 0000		10 = delete
		3000 0000		11 = delete
4-5		0C00 0000	Reserved	
6	L	0200 0000	Load Task	
7			Reserved	
8	H	0080 0000	Task resident	
9-10			Reserved	
11	M	0010 0000	Send Message	
12	Q	0008 0000	Add parameter to specified task's Task Queue	
13	P	0004 0000	Change priority of specified task	
14-15			Reserved	
16	O	0000 8000	Connect specified device to specified task.	
17	T	0000 4000	Thaw: enable interrupts on specified device.	
18	I	0000 2000	SINT. Simulate interrupt on specified device.	
19	F	0000 1000	Freeze: disable interrupts on specified device.	
20	U	0000 0800	Unconnect· disconnect specified device from specified task.	
21-24			Reserved	
25	N	0000 0040	Task Non-resident	
26-28			Reserved	
29-30	A	0000 0000	Start task·	00 = no function requested
		0000 0002		01 = start immediately
		0000 0004		10 = delay start
		0000 0006		11 = delay start
31			Reserved	

SVC 6 Error Codes

The format of the Error Status halfword is as follows:

```
     BIT
        0                    7  8                   15
FIELD  ┌──────────────────────┬──────────────────────┐
       │      POS  PTR         │     ERROR CODE       │
       └──────────────────────┴──────────────────────┘
```

Code Dec (Hex)	Function	Meaning
0(0)	All	No errors; all requested functions complete.
1(1)	All	Syntax error in TASK ID field. Does not apply to self-directed calls
2(2)	— —	Illegal function code.
3(3)	L	Task already present.
4(4)	All but L	No such task in foreground.
5(5)	P	Invalid priority.
6(6)	L	Floating point not supported by SYSGEN
7(7)	A	Specified task not dormant.
8(8)	S	Invalid start address.
10(A)	A (delay)	Invalid code in TT field.
11(B)	M	Message not sent, task unable to receive
12(C)	Q	No queue, full queue, or entries disabled.
13(D)	O,T,I,F,U	No such device in system.
14(E)	O,T,I,F,U	Device named is not a connectable device.
15(F)	O	Device is busy, cannot connect.
16(10)	T,I,F,U	Device not connected to specified task.
17(11)	L	Invalid or unassigned Load LU.
18(12)	L	Checksum error.
25(19)	I	Device is not SINTable.
65(41)	L	Partition does not have sufficient number of LUs.
66(42)	L	Resident Library or Task Common not present.
68(44)	L	Invalid format on Loader Information Block.
73(49)	L	Partition not vacant.
74(4A)	L	I/O error during Roll In/out
75(4B)	L	Assign/close error during Roll In/out
76(4C)	L	Partition does not exist.
129-192 (81-C0)	L	I/O error reading Load LU; error status is as returned by SVC 1.

Condition Code

Unchanged

.

Command/Modifier
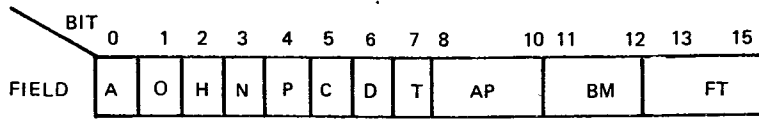
Bits	Name	Hex Mask	Description
0	A	8000	Allocate — requires file type (FT) modifier
1	O	4000	Assign — requires access privilege (AP) and buffer management (BM) modifiers
2	H	2000	Change Access Privilege — requires access privilege (AP) modifiers
3	N	1000	Rename — no modifiers
4	P	0800	Reprotect — no modifiers
5	C	0400	Close — no modifiers
6	D	0200	Delete — no modifiers
7	T	0100	Checkpoint — no modifiers

Bits 0-7 all Zero = Fetch Attributes

Bits	Name		Description
8-10	AP		Access Privileges
		0000	SRO — Shared Read Only
		0020	ERO — Exclusive Read Only
		0040	SWO — Shared Write Only
		0060	EWO — Exclusive Write Only
		0080	SRW — Shared Read-Write
		00A0	SREW — Shared Read-Exclusive Write
		00C0	ERSW — Exclusive Read-Shared Write
		00E0	ERW — Exclusive Read-Write
11-12	BM		Buffer Management
		0000	Default Buffer Management
		0008	Unbuffered (Physical)
		0010	Buffered (Logical)
		0018	Reserved
13-15	FT		File Type
		0000	Contiguous
		0001	Reserved (considered illegal)
		0002	Indexed
		0003	
		....	Reserved (considered illegal)
		0007	

ERROR CODE DEC (HEX)	FUNCTION	MEANING
0 (00)	ALL	No error, the requested functions are complete.
1 (01)	A,O,D	Illegal function, illegal file type
2 (02)	All but A,D	LU error; illegal LU
3 (03)	A,O,D	Volume error; no such volume/device in system.
4 (04)	A,O,N,D	Name error; mismatch on filename or extension field.
5 (05)	A	Size error; erroneous LRECL or size field
6(06)	O,P,D	Protect error; erroneous protection keys.
7(07)	All but C,T,F	Privilege error; unable to obtain requested privilege
8 (08)	O	Buffer error; no room in system for file control blocks or buffers.
9 (09)	All but A, O	Assignment error; LU not assigned.
10 (0A)	A,O,N.P,D	Type error; non-direct access device, or device off-line.
11 (0B)	A,O,N,D	File Descriptor error; illegal syntax.
12 (0C)	O	Attempt to assign SVC 6 connectable device.
129-192 (81-C0)	All but F	I/O error; interpreted as SVC 1 status byte.

SVC 7 - FILE MANAGEMENT SERVICES

CAL CODE

```
         ALIGN  2
PARBLK   DB     CMD
         DB     MOD
         DS     1
         DB     LU
         DB     WKEY
         DB     RKEY
         DC     H'LRECL'
         DC     C'VOLN'
         DC     C'FILENAME'
         DC     C'EXT'
         DC     F'SIZE'
```

FORMAT

0(0) CMD	1(1) MOD
2(2) STAT	3(3) LU
4(4) WKEY	5(5) RKEY
6(6) LRECL	
8(8) VOLN	
12(c) FILENAME	
20(14) EXT	
	23(12) RESERVED
24(18) SIZE	

FUNCTION   REQUIRED FIELDS

	CMD	AP	BM	FT	LU	KEYS	LRECL	VOLN	FILEN	EXT	SIZE
ALLOCATE	✓			✓		✓	✓	✓	✓	✓	✓
ASSIGN	✓	✓	✓		✓	✓		✓	✓	✓	
CHANGE ACCESS PRIVILEGE	✓	✓			✓						
RENAME	✓				✓				✓	✓	
REPROTECT	✓				✓	✓					
CLOSE	✓				✓						
DELETE	✓					✓		✓	✓	✓	
CHECKPOINT	✓		.		✓						
FETCH ATTRIBUTES	✓				✓						

> SVC 9 - LOAD TSW

No parameter block; SVC effective address specifies address of TSW to be loaded.

APPENDIX 2
CONTROL BLOCK SUMMARY

TASK STATUS WORD (TSW)

Task Status Word is 32 Bits. First halfword (bits 0-15) contains status defined later Second halfword (bits 16-31) contains the current location counter as in a PSW.

Bit	Hex Mask	Name	Description
0	8000	W	Trap Wait—task suspended until a task trap occurs
1	4000	P	Power Restoration Trap Enable—a task trap occurs on completion of system power restore sequence.
2-3			Reserved—must be ZERO.
4	0800	Q	Task Queue Service Trap Enable—a task trap occurs when an item is added to the Task Queue. Also a trap occurs if a TSW is loaded with this bit set and the Task Queue is not empty.
5			Reserved—must be ZERO.
6	0200	D	Enable Task Queue entry on interrupt from a connected Trap Generating Device.
7	0100	T	Enable Task Queue entry on Queue Parameter (SVC 6) request directed at this task.
8	0080	Z	Enable Task Queue entry on completion of a time interval.
9	0040	O	Enable Task Queue entry on completion of I/O Proceed request.
10			Reserved—must be ZERO.
11	0010	E	Enable Task Queue entry on task message
12-15	000F	CC	Condition Code (as in PSW).

TASK OPTIONS BITS (TCB)

Bit	Hex Mask	Meaning
0	8000	Executive Task (E-Task) if 1 User Task (U-Task) if 0
1	4000	Arithmetic Fault Continue if 1 Arithmetic Fault Pause if 0
2	2000	Task uses SPFP registers if 1 Task does not use SPFP registers if 0
3	1000	Task is Memory Resident if 1 Task is nonMemory Resident if 0
4	0800	Task uses old SVC 1 format if 1 Task uses OS/16 MT2 SVC 1 format if 0
5	0400	Task is the background task if 1 Task is a foreground task if 0

Bit	Hex Mask	Meaning
6	0200	SVC 6 ignored (Background only) if 1
		SVC 6 illegal (Background only) if 0
8	0100	Task is rollable if 1
		Task is nonrollable if 0
		Reserved for system use
9	0040	Task uses DPFP registers if 1
		Task does not use DPFP registers if 0
10-15		Reserved for system use

TASK STATUS BITS (TCB)

Bit	Hex Mask	Meaning
0	8000	Dormant
1	4000	I/O Wait
2		Reserved
3	1000	Roll Wait
4		Reserved
5	0400	Console Wait (PAUSEd)
6	0200	Time Wait
7	0100	Trap Wait
8	0080	SVC 2 Wait
9	0040	SVC 5/6 Wait
10	0020	SVC 7 Wait
11		Reserved
12	0008	DCB Wait (File I/O Wait)

User Dedicated Locations (UDL)

0 (0)	2 (2)
CTOP	COMBOT
4 (4)	6 (6)
UTOP	UBOT
8 (8)	
A(TASKQ)	

10(A)
POWER RESTORE OLD TSW SAVE AREA
14(E)
POWER RESTORE NEW TSW
18(12)
TASK QUEUE SERVICE OLD TSW SAVE AREA
22(16)
TASK QUEUE SERVICE NEW TSW

26(1A)	28(1C)
A(MESSAGE BUFFER)	RESERVED

26(1A)
RESERVED
36 (24)   SINGLE PRECISION FLOATING POINT REGISTER SAVE AREA
68(44)   DDUBLE PRECISION FLOATING POINT REGISTER SAVE AREA

132(84)

PCBMT2 - OS/16 MT2 PARAMETERS AND CONTROL BLOCKS   PAGE   1

PROG= PCBMT2   ASSEMBLED BY CAL/16 00-00

2   COPY   SGN

0000R

DEFAULT SYSGEN PARAMETERS

```
                    SGN - DEFAULT SYSGEN PARAMETERS
 4          *
 5          *
 6          *
 7  0000    BISYNC   EQU   0     NO BISYNC SUPPORT                              PCB00030
 8  0001    BULKSTOR EQU   1     BULK DEVICE COMMANDS INCLUDED                  PCB00040
 9  0000    CLOCK    EQU   0     NO LINE FREQUENCY CLOCK SUPPORT                PCB00050
10  0040    CMDLEN   EQU   64    INPUT BUFFER LENGTH                            PCB00060
11  0001    CMPROC   EQU   1     INCLUDE COMMAND PROCESSOR              EN003   PCB00070
12  0010    CONS     EQU   16    DEVICE CODE FOR CONSOLE                        PCB00080
13  0002    CONSOLE  EQU   2     DEVICE # FOR CONSOLE                           PCB00090
14  0000    CONSPT   EQU   0     NO CONSOLE PAPER TAPE SUPPORT                  PCB00095
15  0000    CSS      EQU   0     NO CSS SUPPORT                                 PCB00100
16  0000    DATE     EQU   0     MM/DD/YY                                       PCB00110
17  0001    DEBUG    EQU   1     INCLUDE EXAMINE/MODIFY/BIAS                    PCB00120
18  0000    DFLOAT   EQU   0     NO DFPP SUPPORT                       EN001    PCB00130
19  0001    DISCMGT  EQU   1     DIRECT ACCESS SUPPORT                          PCB00140
20  0001    DISP     EQU   1     INCLUDE DISPLAY COMMAND                        PCB00150
21  0000    EFLOAT   EQU   0     NO EXTENDED SPFP TRAPS                EN001    PCB00155
22  0000    FLOAT    EQU   0     NO FLOATING POINT SUPPORT                      PCB00160
23  0000    FOREGRND EQU   0     NO FOREGROUND PARTITIONS                       PCB00170
24  003C    FREQ     EQU   60    60 CPS LINE FREQUENCY CLOCK                    PCB00175
25  0000    INDEX    EQU   0     NO INDEXED FILE SUPPORT                        PCB00180
26  0001    INITIAL  EQU   1     KEEP SYSINIT                                   PCB00190
27  0000    INTERVAL EQU   0     NO PIC INTERVAL                                PCB00200
28  0000    JOURNALO EQU   0     NO SYSTEM JOURNAL                              PCB00210
29  0001    LOBG     EQU   1     INCLUDE BACKGROUND LOADER                      PCB00220
30  0001    LOAD     EQU   1     INCLUDE LOAD COMMAND                           PCB00230
31  0030    LOGLEN   EQU   48    OUTPUT BUFFER LENGTH                           PCB00240
32  0003    LOGQUE   EQU   3     LOG MESSAGE QUEUE SIZE                         PCB00250
33  0000    MEMPROT  EQU   0     NO MEMORY PROTECT                              PCB00260
34  0001    MULOV    EQU   1     INCLUDE MULTIPLY/DIVIDE TRAPS         EN003    PCB00270
35  0001    OPTIONS  EQU   1     INCLUDE OPTION COMMAND                         PCB00280
36  0000    OVCMD    EQU   0     DO NOT OVERLAY COMMAND PROCESSOR               PCB00290
37  0000    OVSEVEN  EQU   0     DO NOT OVERLAY SVC 7                           PCB00295
38  0000    OVSIX    EQU   0     DO NOT OVERLAY SVC 6                           PCB00300
39  0000    OVTWO    EQU   0     DO NOT OVERLAY SVC 2                           PCB00310
40  0000    PCLOCK   EQU   0     NO PIC SUPPORT                                 PCB00320
41  0001    PFREC    EQU   1     INCLUDE POWER FAILURE RECOVERY       EN003    PCB00330
42  0000    PROC     EQU   0     NO LIST INSTRUCTION OR MUL/DIV TRAPS           PCB00340
43  0000    ROLL     EQU   0     NO ROLL IN/ROLL OUT                            PCB00350
44  0000    ROUND    EQU   0     NO ROUND ROBIN SCHEDULING                      PCB00355
45  0000    SAFE     EQU   0     NO SAFETY CHECKS                               PCB00360
46  0001    SETLOG   EQU   1     INCLUDE SET LOG COMMAND                        PCB00370
47  0001    SETPAR   EQU   1     INCLUDE SET PARTITION COMMAND                  PCB00380
48  0001    SVCFIVE  EQU   1     INCLUDE SVC 5                                  PCB00390
49  0001    SVCSEVEN EQU   1     INCLUDE FILE MANAGER                  EN003    PCB00400
50  0001    SVCSIX   EQU   1     INCLUDE SVC 6                                  PCB00410
51  0001    SVCTWO   EQU   1     INCLUDE SVC 2                        EN003    PCB00420
52  000A    UNITS    EQU   10    # OF LOGICAL UNITS PER TASK                    PCB00425
                                                                               PCB00430
54          COPY  REGS                                                         PCB00435
                                                                               PCB00440
```

0000R

REGISTER EQUATES

```
                                                                    PCB008480
56  *                                                               PCB008490
57  *      REGS - REGISTER EQUATES                                  PCB008500
58  *                                                               PCB008510
59  PSWCCCL  EQU  X'FFF0'    CONDITION CODE CLEAR     FFF0           PCB008520
60  PSWCCC   EQU  X'0008'    CARRY FLAG MASK          0008           PCB008530
61  PSWCCV   EQU  X'0004'    V FLAG MASK              0004           PCB008540
62  PSWCCG   EQU  X'0002'    G FLAG MASK              0002           PCB008550
63  PSWCCL   EQU  X'0001'    L FLAG MASK              0001           PCB008560
64  *                                                               PCB008570
65  CRASH    EQU  0                                   0000           PCB008580
66  *                                                               PCB008590
67  R0       EQU  0                                   0000           PCB008600
68  R1       EQU  1                                   0001           PCB008610
69  R2       EQU  2                                   0002           PCB008620
70  R3       EQU  3                                   0003           PCB008630
71  R4       EQU  4                                   0004           PCB008640
72  R5       EQU  5                                   0005           PCB008650
73  R6       EQU  6                                   0006           PCB008660
74  R7       EQU  7                                   0007           PCB008670
75  R8       EQU  8                                   0008           PCB008680
76  R9       EQU  9                                   0009           PCB008690
77  R10      EQU  10                                  000A           PCB008700
78  R11      EQU  11                                  000B           PCB008710
79  R12      EQU  12                                  000C           PCB008720
80  R13      EQU  13                                  000D           PCB008730
81  R14      EQU  14                                  000E           PCB008740
82  R15      EQU  15                                  000F           PCB008750
83  RA       EQU  10                                  000A           PCB008760
84  RB       EQU  11                                  000B           PCB008770
85  RC       EQU  12                                  000C           PCB008780
86  RD       EQU  13                                  000D           PCB008790
87  RE       EQU  14                                  000E           PCB008800
88  RF       EQU  15                                  000F           PCB008810
89  *                                                               PCB008820
90  UTSTAT   EQU  X'7F00'    EI+MM+DF+AS+FP+QS+PM     7F00           PCB008830
91  ETSTAT   EQU  X'7E00'    EI+MM+DF+AS+FP+QS        7E00           PCB008840
92  SUSTAT   EQU  X'7C00'    EI+MM+DF+AS+FP           7C00           PCB008850
93  ISSTAT   EQU  X'3000'    MM+DF                    3000           PCB008860
94  WAITSTAT EQU  X'FE00'    WT+EI+MM+DF+AS+FP+QS     FE00           PCB008870
95  CRSTAT   EQU  X'A000'    WT+MM                    A000
97           COPY  SPT
```

0000R

```
                                                                    PCB00910
                *                                                   PCB00920
                * SPT - SYSTEM POINTER TABLE                        PCB00930
                *                                                   PCB00940
         SPT    STRUC                                               PCB00950
0000     SPT.INIT  DS   12    BRANCH TO SYSINIT                     PCB00960
000C     SPT.CRSH  DS   2     CRASH CODE                            PCB00970
000E     SPT.CE    DS   2     COMMAND ENTRY                         PCB00980
0010     SPT.UBOT  DS   2     USER BOTTOM                           PCB00990
0012     SPT.FBOT  DS   2     FCB BOTTOM                            PCB01000
0014     SPT.MTOP  DS   2     MACHINE TOP                           PCB01010
0016     SPT.OSID  DS   8     NAME OF OS                            PCB01020
001E     SPT.TTAB  DS   2     TCB TABLE                             PCB01030
0020     SPT.NTCB  DS   1     # OF TCBS                             PCB01040
0021     SPT.CTCB  DS   1     CURRENT TCB INDEX                     PCB01050
0022     SPT.DMT   DS   2     DMT ADDR                              PCB01060
0024     SPT.VMT   DS   2     VMT ADDR                              PCB01070
0026     SPT.JRNL  DS   2     JOURNAL LIST ADDRESS                  PCB01080
0028     SPT.FLBA  DS   4     FLBA OF OS OVERLAY FILE               PCB01090
002C     SPT.OVFD  DS   16    NAME OF OS OVERLAY FILE               PCB01100
003C     SPT.SDCB  DS   2     POINTER TO DCB OF OS OVERLAY DEVICE   PCB01110
003E     ADR.SPT   EQU  X'62'  A(A(SPT))                           PCB01120
  0062            ENDS

0000R           COPY  TCB
```

APPENDIX 3 (Continued)

LOC	VALUE	LINE	SOURCE	COMMENT	SEQ
		124	*		PCB01160
		125	* TCB - TASK CONTROL BLOCK		PCB01170
		126	*		PCB01180
		127	TCB STRUC		PCB01190
0000		128	TCB.ID DS 8	TASK IO	PCB01200
0008		129	TCB.STAT DS 2	TASK STATUS WORD	PCB01210
000A		130	TCB.ITSW DS 4	INITIAL PSW AND LOC	PCB01220
000E		131	TCB.CPSW DS 4	CURRENT PSW	PCB01230
0012		132	TCB.CTSW DS 2	CURRENT TASK STATUS HALFWORD	PCB01240
0014		133	TCB.UBOT DS 2	A(PARTITION)	PCB01250
0016		134	TCB.MPRI DS 1	MAXIMUM PRIORITY	PCB01260
0017		135	TCB.CPRI DS 1	CURRENT PRIORITY	PCB01270
0018		136	DS 1	RESERVED	PCB01280
0019		137	TCB.WTHR DS 1	INDEX OF NEXT TCB IN WAIT THREAD	PCB01290
001A		138	TCB.TVAL DS 4	TIME INTERVAL WAIT	PCB01300
001E		139	TCB.ADQT DS 4	TIME INTERVAL FOR TASK QUEUE	PCB01310
0022		140	TCB.ADQP DS 2	TASK QUEUE PARAMETER	PCB01320
0024		141	TCB.OPT DS 2	TASK OPTIONS	EN001 PCB01330
0026		142	TCB.FLGS DS 2	FLAGS	EN001 PCB01340
0028		143	TCB.WTAD DS 2	A(WAIT THREAD)	PCB01350
002A		144	TCB.MPT DS 8	MEMORY PROTECT PATTERN	PCB01360
0032		145	TCB.NLU DS 1	# OF LOGICAL UNITS	PCB01370
0033		146	TCB.PNT DS 1	TCB TABLE POINTER	PCB01380
0034		147	TCB.RC DS 2	RETURN CODE	PCB01390
0036		148	TCB.RSAV DS 32	DISPATCH REGISTER SAVE AREA	PCB01400
0056		149	TCB.FMLU DS 2	FILE MANAGER LU	PCB01410
0058		150	TCB.LTAB EQU *	LOGICAL UNIT TABLE	PCB01420
		151	* TASK STATUS		PCB01430
	8000	152	TSTT.DMM EQU X'8000'	DORMANT	PCB01440
	4000	153	TSTT.IOM EQU X'4000'	I/O WAIT	PCB01450
	1000	154	TSTT.RWM EQU X'1000'	ROLL WAIT	PCB01460
	0400	155	TSTT.CWM EQU X'0400'	CONSOLE WAIT (PAUSED)	PCB01470
	0200	156	TSTT.TIM EQU X'0200'	TIME WAIT	PCB01480
	0100	157	TSTT.TRM EQU X'0100'	TRAP WAIT	PCB01490
	0080	158	TSTT.S2M EQU X'0080'	SVC 2 WAIT	PCB01500
	0040	159	TSTT.S5M EQU X'0040'	SVC 5 OR 6 WAIT	PCB01510
	0020	160	TSTT.S7M EQU X'0020'	SVC 7 WAIT	PCB01520
	0008	161	TSTT.DWM EQU X'0008'	DCB WAIT	PCB01530
		162	* TASK OPTIONS/FLAGS MASKS		PCB01540
	8000	163	TOPT.ETM EQU X'8000'	ET STATE	PCB01550
	4000	164	TOPT.ACM EQU X'4000'	ARITHMETIC FAULT CONTINUE	PCB01560
	2000	165	TOPT.FPM EQU X'2000'	FLOATING POINT	PCB01570
	1000	166	TOPT.MRM EQU X'1000'	MEMORY RESIDENT	PCB01580
	0800	167	TOPT.COM EQU X'0800'	COMPATIBILITY	PCB01590
	0400	168	TOPT.BGM EQU X'0400'	BACKGROUND TASK	PCB01595
	0200	169	TOPT.S6M EQU X'0200'	SVC 6 CONTINUE	PCB01600
	0040	170	TOPT.DFM EQU X'0040'	DOUBLE FLOATING POINT	PCB01605
	0100	171	TOPT.ROM EQU X'0100'	ROLLABLE	EN001 PCB01610
		172	*		PCB01620
	0080	173	TFLG.PPM EQU X'0080'	PAUSE PENDING FLAG	PCB01630
	0040	174	TFLG.CPM EQU X'0040'	CANCEL PENDING	PCB01640
	0020	175	TFLG.SLM EQU X'0020'	INSIDE SYSTEM LEVEL CODE	PCB01660
	0010	176	TFLG.BIM EQU X'0010'	INSIDE BISYNC LEVEL CODE	PCB01670
	0008	177	TFLG.PRM EQU X'0008'	POWER RESTORE TRAP PENDING	PCB01680
	0004	178	TFLG.ROM EQU X'0004'	ROLL REQUEST PENDING	PCB01690
	0002	179	TFLG.TOD EQU X'0002'	TIME OF DAY FLAG	PCB01695
	0001	180	TFLG.TPM EQU X'0001'	TRAP PENDING FLAG	

APPENDIX 3 (Continued)

TASK CONTROL BLOCK

Value	Line	Label	Op	Operand	Comment	Ref
4000	183	TTSW.PRM	EQU	X'4000'	POWER RESTORE TRAP ENABLE	PCB01720
0800	184	TTSW.QSM	EQU	X'0800'	QUEUE SERVICE TRAP ENABLE	PCB01730
0200	185	TTSW.DIM	EQU	X'0200'	QUEUE DEVICE INTERRUPT	PCB01740
0100	186	TTSW.TCM	EQU	X'0100'	QUEUE TASK CALL	PCB01750
0080	187	TTSW.TOM	EQU	X'0080'	QUEUE TIMEOUT	PCB01760
0040	188	TTSW.IPM	EQU	X'0040'	QUEUE I/O PROCEED COMPLETE	PCB01770
0010	189	TTSW.PMM	EQU	X'0010'	QUEUE SEND MESSAGE	EN006 PCB01775
	190	* LU TABLE ATTRIBUTE BYTE				PCB01780
0040	191	TLAT.RDM	EQU	X'40'	READ	PCB01790
0020	192	TLAT.WRM	EQU	X'20'	WRITE	PCB01800
0010	193	TLAT.BIM	EQU	X'10'	BINARY	PCB01810
0008	194	TLAT.WTM	EQU	X'08'	WAIT	PCB01820
0004	195	TLAT.RNM	EQU	X'04'	RANDOM	PCB01830
0002	196	TLAT.UPM	EQU	X'02'	UNCONDITIONAL PROCEED	PCB01840
0001	197	TLAT.IMM	EQU	X'01'	IMAGE	PCB01850
0058	198		ENDS			PCB01860
0000R	200		COPY	DCB		

```
                  OCB - DEVICE CONTROL BLOCK

202          *
203          *
204          *        OCB - DEVICE CONTROL BLOCK
205 FFEA  DCB.RECL EQU  -22       LOGICAL RECORD LENGTH            -2B      PCB01900
206 FFEC  DCB.CNTS EQU  -20       COUNTS                          -2 BYTES PCB01910
207 FFEC  DCB.WCNT EQU  -20       WRITE COUNT                     -1B      PCB01920
208 FFED  DCB.RCNT EQU  -19       READ COUNT                      -1B      PCB01930
209 FFEE  OCB.DCOD EQU  -18       DEVICE CODE                     -1B      PCB01940
210 FFEF  DCB.DN   EQU  -1T       DEVICE #                        -1B      PCB01950
211 FFF0  DCB.ATRB EQU  -16       DEVICE ATTRIBUTES               -2B      PCB01960
212 FFF2  DCB.KEYS EQU  -14       KEYS                            -2 BYTES PCB01970
213 FFF2  DCB.WKEY EQU  -14       WRITE KEY                       -1B      PCB01980
214 FFF3  DCB.RKEY EQU  -13       READ KEY                        -1B      PCB01990
215 FFF4  DCB.BUSY EQU  -12       A(BUSY FLAG)                    -2 BYTES PCB02000
216 FFF4  DCB.SBSY EQU  -12       A(SELCH BUSY FLAG)              -2B      PCB02010
217 FFF6  DCB.TERM EQU  -10       A(TERMINATION ROUTINE)          -2B      PCB02020
218 FFF8  DCB.ALOC EQU  -8        ABORT LOC                       -2B      PCB02030
219 FFFA  DCB.TOUT EQU  -6        TIME OUT COUNT(0=TIMED OUT      -2B      PCB02040
220 FFFC  DCB.FLGS EQU  -4        FLAGS HALFWORD                  -2B      PCB02050
221 FFFE  DCB.INIT EQU  -2        A(INITIALIZATION ROUTINE)       -2B      PCB02060
222          STRUC                                                        PCB02070
223 0000  DCB.OPSW DS   4         OLD PSW SAVE AREA                        PCB02080
224 0004  DCB.NPSS DS   2         NEW PSW STATUS                           PCB02090
225 0006  DCB.ENTR DS   10        INTERRUPT SERVICE ENTRY POINT            PCB02100
226 0010  DCB.LEAV DS   4         INTERRUPT SERVICE EXIT LOCATION          PCB02110
227 0014  DCB.NOPI DS   8         NOP ISR ENTRY                            PCB02120
228 001D  DCB.IOW  EQU  *+1       I/O WAIT POINTER                -1B      PCB02130
229 001C  DCB.RSAV DS   14*2      SAVE AREA FOR REGISTERS 2-15             PCB02140
230 0036  DCB.ISR  EQU  *-2       ISR POINTER (R15)               -2B      PCB02150
231 0038  DCB.RES  DS   2         RESERVED FOR BUSY FLAG                   PCB02160
232 003A  DCB.EDDC EQU  *         BISYNC EXTENDED DEVICE CODE     -2 BYTES PCB02170
233 003A  DCB.SDN  DS   1         SELCH DEVICE #                           PCB02180
234 003B  DCB.CDN  DS   1         CONTROLLER DEVICE #                      PCB02190
235 003C  DCB.CLRT EQU  *         A(CLOSE ROUTINE) FOR ITAM DRIVERS        PCB02200
236 003C  DCB.DBSY DS   2         A(DISC BUSY FLAG)                        PCB02210
237 003E  DCB.BFSZ EQU  *         BISYNC BUFFER SIZE              -2 BYTES PCB02220
238 003E  DCB.DPB  EQU  *         DUMMY PARAMETER BLOCK                    PCB02230
239 0040  DCB.FCB  EQU  *+2       A(FCB) SAVE AREA                -2B      PCB02240
240 003E  DCB      DS   14                                                 PCB02250
241 004C  DCB.FMEX DS   2         FILE MANAGER ALTERNATE EXIT     EN004    PCB02260
242 004E  DCB.CSEC DS   4         CURRENT SECTOR POINTER          EN010    PCB02270
243 0052  DCB.DIRP DS   4         FIRST DIRECTORY BLOCK POINTER-VD,FDP     PCB02280
244 0056  DCB.BITP DS   4         BIT MAP POINTER-VD.MAP                   PCB02290
245 005A  DCB.CCYL DS   2         CURRENT CYLINDER POINTER                 PCB02300
246 005C  DCB.FMEP DS   4         FILE MANAGER ENTRY POINT                 PCB02310
247 005E  DCB.SCYL DS   2         NUMBER OF SECTORS/CYLINDER               PCB02320
248 0060  DCB.STRK DS   2         NUMBER OF SECTORS/TRACK                  PCB02332
249 0061  DCB.OINC DS   1         OPTIMUM INCREMENT                        PCB02334
250 0062  DCB.SIZE DS   4         # OF SECTORS ON DEVICE                   PCB02336
251 0066  DCB.ASAV DS   16*2      ALTERNATE SAVE AREA                      PCB02338
252 0086  DCB.FSAV DS   4         FULLWORD SAVE AREA                       PCB02340
253 008A  DCB.BINX DS   2         CURRENT INDEX INTO BIT MAP BUFFER        PCB02350
254 008C  DCB.CBIT DS   4         CURRENT BIT MAP SECTOR POINTER           PCB02360
255 0090  DCB.BUFB DS   256       BIT MAP SECTOR                           PCB02370
256 01A6  DCB      EQU  *=DCB.RECL                                         PCB02380
257          *        FLAGS MASKS                                         PCB02390
258 8000  DFLG.BLM EQU  X'8000'   BULK DEVICE FLAG                         PCB02410
259 4000  DFLG.LNM EQU  X'4000'   ON-LINE FLAG                             PCB02420
```

```
0000  1000   261  DFLG.MPP  EQU   X'1000'   BIT MAP PRESENCE                      PCB02450
      0800   262  DFLG.PFM  EQU   X'0800'   PSUDO FILEMARK CHECK                  PCB02460
      0400   263  DFLG.BMM  EQU   X'0400'   BIT MAP MODIFY FLAG                   PCB02470
      0200   264  DFLG.CNM  EQU   X'0200'   CONSOLE FLAG                          PCB02480
      0100   265  DFLG.BTM  EQU   X'0100'   BISYNC DEVICE FLAG                    PCB02490
      0080   266  DFLG.SBF  EQU   X'0080'   SVC 6 CONNECTABLE DEVICE              PCB02500
      0040   267  DFLG.WPM  EQU   X'0040'   WRITE PROTECT FLAG                    PCB02510
      0020   268  DFLG.PIM  EQU   X'0020'   HALT I/O ABORT FLAG            END09   PCB02520
      0010   269  DFLG.SIM  EQU   X'0010'   FMCR CALL FROM SVC1 FLAG              PCB02530
      0008   270  DFLG.SOM  EQU   X'0008'   START (A) ZERO FLAG (SET SFC)         PCB02540
      0004   271  DFLG.IUM  EQU   X'0004'   DCB IN USE FLAG                       PCB02550
      0001   272  DFLG.SDM  EQU   X'0001'   SELCH DEVICE FLAG              END04   PCB02550
             273  *               ATTRIBUTES MASKS                               PCB02560
      8000   274  DATR.INM  EQU   X'8000'   INTERACTIVE DEVICE            END09   PCB02565
      4000   275  DATR.RDM  EQU   X'4000'   SUPPORTS READ                         PCB02570
      2000   276  DATR.URM  EQU   X'2000'   SUPPORTS WRITE                        PCB02580
      1000   277  DATR.BIM  EQU   X'1000'   SUPPORTS BINARY                       PCB02590
      0800   278  DATR.WTM  EQU   X'0800'   SUPPORTS WAIT I/O                     PCB02600
      0400   279  DATR.RNM  EQU   X'0400'   SUPPORTS RANDOM                       PCB02610
      0200   280  DATR.UPM  EQU   X'0200'   SUPPORTS UNCONDITIONAL PROCEED        PCB02620
      0100   281  DATR.IMM  EQU   X'0100'   SUPPORTS IMAGE                        PCB02630
      0080   282  DATR.HTM  EQU   X'0080'   SUPPORTS HALT I/O             END09   PCB02635
      0040   283  DATR.RWM  EQU   X'0040'   SUPPORTS REWIND                       PCB02640
      0020   284  DATR.BRM  EQU   X'0020'   SUPPORTS BACKSPACE RECORD             PCB02650
      0010   285  DATR.FRM  EQU   X'0010'   SUPPORTS FORWARD SPACE RECORD         PCB02660
      0008   286  DATR.NFM  EQU   X'0008'   SUPPORTS WRITE FILEMARK               PCB02670
      0004   287  DATR.FFM  EQU   X'0004'   SUPPORTS FORWARD SPACE FILEMARK       PCB02680
      0002   288  DATR.BFM  EQU   X'0002'   SUPPORTS BACKSPACE FILEMARK           PCB02690
      00FF   289  DCDD.NUL  EQU   255       DEVICE CODE OF NULL DEVICE            PCB02700
             290            ENDS                                                  PCB02710

0190         292            COPY  DMT.                                            PCB02740
             293  *                                                               PCB02750
             294  *         DMT - DEVICE MNEMONIC TABLE                           PCB02760
             295  *                                                               PCB02770
0000R        296  DMT.      STRUC                                                 PCB02780
0000         297  DMT.NAME  DS    4         DEVICE MNEMONIC                       PCB02790
0004         298  DMT.DCB   DS    ADC       A(DCB)                                PCB02800
0006         299            ENDS

0000R        301            COPY  VMT.                                            PCB02830
             302  *                                                               PCB02840
             303  *         VMT - VOLUME MNEMONIC TABLE                           PCB02850
             304  *                                                               PCB02860
0000R        305  VMT.      STRUC                                                 PCB02870
0000         306  VMT.VOLN  DS    4         VOLUME NAME                           PCB02880
0004         307  VMT.DMT   DS    ADC       A(DMT ENTRY)                          PCB02890
0006         308            ENDS

0000R        310            COPY  FCB
```

APPENDIX 3 (Continued)

```
0000                    *
                        *  FCB - FILE CONTROL BLOCK
                        *
                        FCBI     STRUC
0000    FCB.VMT   DS    2        ADDRESS OF VMT ENTRY        -2 BYTES    PCB02950
0002    FCB.CNTS  DS    0        COUNTS                                  PCB02940
0002    FCB.WCNT  DS    1        WRITE COUNT                             PCB02930
0003    FCB.RCNT  DS    1        READ COUNT                              PCB02960
0004    FCB.ATRB  DS    2        ATTRIBUTES OF FILE                      PCB02970
0006    FCB.LRCL  DS    2        LOGICAL RECORD LENGTH                   PCB02980
0008    FCB.OFF   DS    1        DIRECTORY OFFSET                        PCB02990
0009    FCB.BKSZ  DS    1        FILE BLOCK SIZE                         PCB03000
000A    FCB.DIR   DS    4        ADDRESS OF DIRECTORY BLOCK              PCB03010
000E    FCB.FLGS  DS    1        FLAGS                                   PCB03020
000F    FCB.DCOD  DS    1        DEVICE CODE OF FILE 2=IN,0=CO           PCB03030
0010    FCB.NAME  DS    8        FILENAME                                PCB03040
0018    FCB.EXT   DS    3        EXTENTION                               PCB03050
001B    FCB.VERS  DS    1        VERSION                                 PCB03060
001C    FCB.DCB   DS    2        ADDRESS OF DCB                          PCB03070
001E    FCB.FLBA  DS    4        FIRST LOGICAL BLOCK ADDRESS             PCB03080
0022    FCB.LLBA  DS    4        LAST LOGICAL BLOCK ADDRESS              PCB03090
0026    FCB.CSEC  DS    4        CURRENT SECTOR/NLR                      PCB03100
002A    FCB.LU    DS    1        LOGICAL UNIT                            PCB03110
002B    FCB.TPNT  DS    1        TCB POINTER                             PCB03120
002C    FCBO      EQU   *        LENGTH OF CONTIGUOUS FCB                PCB03130
                        *  ADDITIONAL FOR INDEXED FILES ONLY            PCB03140
002C    FCB.BLK   DS    2        BKSZ IN BYTES                           PCB03150
002E    FCB.IBLK  DS    2        INBS IN BYTES                           PCB03160
0030    FCB.INBS  DS    1        INDEX BLOCK SIZE                        PCB03170
0031    FCB.CCR   DS    1        CLEAR CHARACTER                         PCB03180
0032    FCB.BAPB  DS    14       DATA BLOCK PARAMETER BLOCK              PCB03190
0040    FCB.IBPB  DS    14       INDEX BLOCK PARAMETER BLOCK             PCB03200
004E    FCB.NINB  DS    2        # OF INDEX BLOCKS                       PCB03210
0050    FCB.CINB  DS    2        CURRENT INDEX BLOCK #                   PCB03220
0052    FCB.CINX  DS    2        CURRENT OFFSET INTO INDEX BLOCK         PCB03230
0054    FCB.NBLK  DS    4        # OF BLOCKS IN FILE                     PCB03240
0058    FCB.CBLK  DS    4        CURRENT BLOCK #                         PCB03250
005C    FCB.CLRL  DS    4        CURRENT LOGICAL RECORD #                PCB03260
0060    FCB.COFF  DS    2        CURRENT DATA BLOCK OFFSET               PCB03270
0062    FCB.RSAV  DS    3*2      SAVE AREA FOR R13-R15                   PCB03280
        FCB.BUFF  EQU   *        START OF INDEXED FILE BUFFER            PCB03290
                        *  FLAGS MASKS                                   PCB03300
0080    FFLG.BAM  EQU   X'80'    BUFFERED ACCESS METHOD                  PCB03310
0040    FFLG.IUM  EQU   X'40'    FCB IN USE FLAG                         PCB03320
0020    FFLG.OPM  EQU   X'20'    OPERATION:SET IMPLIES WRITE             PCB03330
0008    FFLG.BMM  EQU   X'08'    DATA MOD FLAG                           PCB03340
0004    FFLG.MOM  EQU   X'04'    MODE:SET IMPLIES EXTENDING FILE         PCB03350
0002    FFLG.SBM  EQU   X'02'    BUSY FLAG SET FLAG                      PCB03360
0001    FFLG.XMM  EQU   X'01'    INDEX MOD FLAG                          PCB03370
                        *  DEVICE CODES                                  PCB03380
0000    FDCD.CO   EQU   0        CONTIGUOUS FILE                         PCB03390
0002    FDCD.IN   EQU   2        INDEXED FILE                            PCB03400
                        *  INDEXED FILE LINK FIELDS                      PCB03410
0000    FINX.PRV  EQU   0        LINK TO PREVIOUS INDEX BLOCK            PCB03420
0004    FINX.NXT  EQU   4        LINK TO NEXT INDEX BLOCK                PCB03430
0068              ENDS                                                   PCB03440
```

SVC1 PARAMETER BLOCK

APPENDIX ? (Continued)

```
                  *        SVC1 - SVC1 PARAMETER BLOCK
                  *
371               *
372      SVC1.    STRUC
373      SVC1.FUN DS    1                    FUNCTION CODE/LOGICAL UNIT
374      SVC1.FC  DS    1                    FUNCTION CODE
375      SVC1.LU  DS    1                    LOGICAL UNIT
376      SVC1.STA DS    1                    STATUS
377      SVC1.DDS DS    1                    DEVICE INDEPENDENT STATUS
378      SVC1.DDS DS    1                    DEVICE DEPENDENT STATUS
379      SVC1.SAD DS    ADC                  STARTING ADDRESS
380      SVC1.EAD DS    ADC                  ENDING ADDRESS
381      SVC1.RAD DS    ADC                  RANDOM ADDRESS
382      SVC1.LXF DS    ADC                  LENGTH OF LAST TRANSFER
383               *
384               *        FUNCTION CODE
385      S1FC.RDR EQU    X*40*               READ
386      S1FC.WRM EQU    X*20*               WRITE
387      S1FC.BIM EQU    X*1B*               BINARY/ASCII
388      S1FC.WTM EQU    X*02*               WAIT/PROCEED
389      S1FC.RNM EQU    X*02*               RANDOM/SEQUENTIAL
390      S1FC.UPM EQU    X*02*               UNCONDITIONAL PROCEED
391      S1FC.IMM EQU    X*04*               IMAGE/FORMATTED
392      S1FC.CMM EQU    X*08*               COMMAND FUNCTION
393      S1FC.RWM EQU    X*40*               REWIND
394      S1FC.BRM EQU    X*20*               BACKSPACE RECORD
395      S1FC.FRM EQU    X*10*               FORWARD SPACE RECORD
396      S1FC.WFM EQU    X*08*               WRITE FILE MARK
397      S1FC.FF  EQU    X*04*               FORWARD FILE
398      S1FC.BFM EQU    X*02*               BACKWARD FILE
399               *
400               *        STATUS HALFWORD
401      S1ST.ERM EQU    X*8000*             ERROR STATUS
402      S1ST.IFM EQU    X*4000*             ILLEGAL FUNCTION
403      S1ST.DUM EQU    X*2000*             DEVICE UNAVAILABLE
404      S1ST.EMM EQU    X*1000*             END OF MEDIUM
405      S1ST.EFM EQU    X*0800*             END OF FILE
406      S1ST.UEM EQU    X*0400*             UNRECOVERABLE ERROR
407      S1ST.REM EQU    X*0200*             RECOVERABLE ERROR
408      S1ST.LUM EQU    X*0100*             ILLEGAL OR UNASSIGNED LU
409               ENDS

411               COPY   SVC5.
```

000E
0000R

29-429 R02 7/76

SVC5 PARAMETER BLOCK

```
                                                                      PCB03930
413   *                                                               PCB03940
414   *                SVC5. - SVC5 PARAMETER BLOCK                    PCB03950
415   *                                                               PCB03960
416        SVC5.      STRUC                                            PCB03970
0000 417   SVC5.ID    DS     8            OVERLAY NAME                 PCB03980
0008 418   SVC5.STA   DS     1            STATUS                       PCB03990
0009 419   SVC5.OPT   DS     1            OPTIONS                      PCB04000
000A 420   SVC5.LU    DS     2            LU NUMBER                    PCB04010
     421   *  OPTION BYTE                                             PCB04020
0004 422   S5OP.RWM   EQU    X'04'        REWIND LU BEFORE LOAD        PCB04030
0001 423   S5OP.NPM   EQU    X'01'        LOAD WITHOUT POSITIONING LU  PCB04040
000C 424              ENDS

0000R 426             COPY   SVC6.
```

```
SVC6 PARAMETER BLOCK

         428  *                                                          PCB04080
         429  *         SVC6. - SVC6 PARAMETER BLOCK                      PCB04090
         430  *                                                          PCB04100
0000     431  SVC6.    STRUC                                             PCB04110
0000     432  SVC6.ID   DS    6         TASK ID                          PCB04120
0006     433  SVC6.FUN  DS    0         FUNCTION CODE      =4 BYTES      PCB04130
0006     434  SVC6.FC1  DS    2         PART 1                           PCB04140
0008     435  SVC6.FC2  DS    2         PART 2                           PCB04150
000A     436  SVC6.TST  DS    2         TASK STATUS                      PCB04160
000C     437  SVC6.STA  DS    2         ERROR POINTER AND CODE           PCB04170
000E     438  SVC6.LU   DS    1         LOAD LU                          PCB04180
000F     439  SVC6.PRI  DS    1         PRIORITY)                        PCB04190
0010     440  SVC6.RPI  DS    1         RETURN PRIORITY                  PCB04200
0011     441           DS    1         RESERVED                         PCB04210
0012     442  SVC6.SAD  DS    ADC       START ADDRESS                    PCB04220
0016     443  SVC6.TIM  DS    0         TIME VALUES:                     PCB04230
0016     444  SVC6.TTM  DS    1         TYPE TIME                        PCB04240
0017     445  SVC6.DTM  DS    3         DELAY TIME                       PCB04250
001A     446  SVC6.DMN  DS    4         DEVICE MNEMONIC                  PCB04260
001E     447  SVC6.PAR  DS    ADC       PARAMETER            EN006       PCB04270
0020     448  SVC6.MSG  DS    ADC       A(MESSAGE)           EN006       PCB04275
0022     449           DS    2*ADC      RESERVED                         PCB04280
8000     450  SFC1.DOM  EQU   X'8000'   DIRECTION:OTHER TASKS            PCB04290
C000     451  SFC1.DSM  EQU   X'C000'   DIRECTION:SELF                   PCB04300
2000     452  SFC1.EDM  EQU   X'2000'   END TASK:DELETE                  PCB04310
1000     453  SFC1.ECM  EQU   X'1000'   END TASK:CANCEL                  PCB04320
0200     454  SFC1.LM   EQU   X'0200'   LOAD TASK                        PCB04330
0080     455  SFC1.FXM  EQU   X'0080'   FIX (MAKE RESIDENT)              PCB04340
0010     456  SFC1.MM   EQU   X'0010'   SEND MESSAGE         EN006       PCB04345
0008     457  SFC1.QM   EQU   X'0008'   QUEUE PARAMETER                  PCB04350
0004     458  SFC1.PM   EQU   X'0004'   CHANGE PRIORITY                  PCB04360
8000     459  SFC2.CM   EQU   X'8000'   CONNECT DEVICE TO SPECIFIED TASK PCB04870
4000     460  SFC2.TM   EQU   X'4000'   THAW  (ENABLE INTERRUPTS)        PCB04380
2000     461  SFC2.IM   EQU   X'2000'   SINT  (SIMULATE INTERRUPT)       PCB04590
1000     462  SFC2.FM   EQU   X'1000'   FREEZE(DISABLE INTERRUPTS)       PCB04400
0800     463  SFC2.UM   EQU   X'0800'   UNCONNECT DEVICE                 PCB04410
0020     464  SFC2.UFM  EQU   X'0020'   UNFIX (MAKE NON-RESIDENT)        PCB04420
0004     465  SFC2.SDM  EQU   X'0004'   DELAY START                      PCB04430
0002     466  SFC2.SIM  EQU   X'0002'   START IMMEDIATELY                PCB04440
         467           ENDS                                             PCB04450

0026     469           COPY  SVC7.
0000R
```

SVC7 PARAMETER BLOCK

```
                *                                                          PCB04490
                *          SVC7. - SVC7 PARAMETER BLOCK                     PCB04500
                *                                                          PCB04510
0000      474   SVC7.     STRUC                                            PCB04520
0000      475   SVC7.OPT  DS   0     OPTION/MODIFIER          -2 BYTES      PCB04530
0000      476   SVC7.CMD  DS   1     COMMAND                                PCB04540
0001      477   SVC7.MOD  DS   1     MODIFIER                               PCB04550
0002      478   SVC7.STA  DS   1     STATUS                                 PCB04560
0003      479   SVC7.LU   DS   1     LOGICAL UNIT                           PCB04570
0004      480   SVC7.KYS  DS   0     KEYS                     -2 BYTES      PCB04580
0004      481   SVC7.WKY  DS   1     WRITE KEY                              PCB04590
0005      482   SVC7.RKY  DS   1     READ KEY                               PCB04600
0006      483   SVC7.LRC  DS   2     LOGICAL RECORD LENGTH                  PCB04610
0008      484   SVC7.VOL  DS   4     VOLUME NAME                            PCB04620
000C      485   SVC7.FNM  DS   8     FILE NAME                              PCB04630
0014      486   SVC7.EXT  DS   3     EXTENSION                              PCB04640
0017      487   SVC7.VER  DS   1     VERSION (NOT USED)                     PCB04650
0018      488   SVC7.SIZ  DS   0     FILE SIZE                -4 BYTES      PCB04660
0018      489   SVC7.ISZ  DS   2     INDEX BLOCK SIZE                       PCB04670
001A      490   SVC7.DSZ  DS   2     DATA BLOCK SIZE                        PCB04680
          491   *                    COMMAND                                PCB04690
8000      492   S7CM.ALM  EQU  X'8000'    ALLOCATE                          PCB04700
4000      493   S7CM.ASN  EQU  X'4000'    ASSIGN                            PCB04710
2000      494   S7CM.CAM  EQU  X'2000'    CHANGE ACCESS PRIVELEGES          PCB04720
1000      495   S7CM.RNM  EQU  X'1000'    RENAME                            PCB04730
0800      496   S7CM.RPM  EQU  X'0800'    REPROTECT                         PCB04740
0400      497   S7CM.CLM  EQU  X'0400'    CLOSE                             PCB04750
0200      498   S7CM.DEM  EQU  X'0200'    DELETE                            PCB04760
0100      499   S7CM.CPM  EQU  X'0100'    CHECKPOINT                        PCB04770
0000      500   S7CM.FEM  EQU  X'0000'    FETCH ATTRIBUTES                  PCB04780
          501   *                    MODIFIER BYTE                          PCB04790
0000      502   S7M.SRO   EQU  X'00'    P                                   PCB04800
0020      503   S7M.ERO   EQU  X'20'    R                                   PCB04810
0040      504   S7M.SWO   EQU  X'40'    I                                   PCB04820
0060      505   S7M.EWO   EQU  X'60'    V                                   PCB04830
0080      506   S7M.SRW   EQU  X'80'    H                                   PCB04840
00A0      507   S7M.SREW  EQU  X'A0'    L                                   PCB04850
00C0      508   S7M.ERSW  EQU  X'C0'    E                                   PCB04860
00E0      509   S7M.ERW   EQU  X'E0'    G                                   PCB04870
          510   *                    BUFFER MANAGEMENT                      PCB04880
0000      511   S7M.DFT   EQU  X'00'    DEFAULT BUFFER MANAGEMENT           PCB04890
0008      512   S7M.UBUF  EQU  X'08'    UNBUFFERED PHYSICAL                 PCB04900
0010      513   S7M.BUF   EQU  X'10'    BUFFERED LOGICAL                    PCB04910
          514   *                    FILE TYPES                             PCB04920
0000      515   S7M.CO    EQU  X'00'    CONTIGUOUS FILE                     PCB04930
0002      516   S7M.IN    EQU  X'02'    INDEXED FILE                        PCB04940
          517             ENDS                                             PCB04950

001C      519             COPY   VD
0000R
```

```
                                                                        PCB04990
521   *                                                                 PCB05000
522   *            VD - VOLUME DESCRIPTOR                               PCB05010
523   *                                                                 PCB05020
0000      524   VD       STRUC                                          PCB05030
0000      525   VD.VOL   DS    4        VOLUME NAME                     PCB05040
0004      526   VD.ATRB  DS    4        VOLUME ATTRIBUTES               PCB05050
0008      527   VD.FDP   DS    4        FIRST DIRECTORY BLOCK POINTER   PCB05060
000C      528   VD.OSP   DS    4        OS IMAGE POINTER                PCB05070
0010      529   VD.OSS   DS    4        OS IMAGE SIZE (IN SECTORS)      PCB05080
0014      530   VD.MAP   DS    4        BIT MAP POINTER                 PCB05090
0018      531            ENDS                                           PCB05095
8000      532   VATR.ONM EQU   X'8000'  DISC IS MARKED ON      457D

0000R     534            COPY  DIR
```