Publication Number 29-390R05

OS/32-MT
PROGRAM REFERENCE MANUAL

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

November 1976

PAGE REVISION STATUS SHEET

PUBLICATION NUMBER 29-390

TITLE OS/32 MT PRM

REVISION R05 DATE November 1976

PAGE	REV.	DATE	PAGE	REV.	DATE	PAGE	REV.	DATE
1- 1	R05	11/76	4-19	R04	4/76	5-27	R04	4/76
1- 2	R04	4/76	4-20	R04	4/76	5-28	R04	4/76
1- 3	R04	4/76	4-21	R04	4/76	5-29	R04	4/76
1- 4	R04	4/76	4-22	R05	11/76	5-30	R04	4/76
1- 5/			4-23	R04	4/76	5-31	R04	4/76
1- 6	R04	4/76	4-24	R04	4/76	5-32	R04	4/76
			4-25	R04	4/76	5-33/		
2- 1	R04	4/76	4-26	R05	11/76	5-34	R04	4/76
2- 2	R04	4/76	4-27	R04	4/76			
			4-28	R04	4/76	A1- 1	R04	4/76
3- 1	R04	4/76	4-29	R04	4/76	A1- 2	R04	4/76
3- 2	R04	4/76	4-30	R04	4/76	A1- 3	R04	4/76
3- 3	R04	4/76	4-31	R05	11/76	A1- 4	R04	4/76
3- 4	R04	4/76	4-32	R04	4/76	A1- 5	R04	4/76
3- 5	R04	4/76	4-33	R04	4/76	A1- 6	R04	4/76
3- 6	R04	4/76	4-34	R05	11/76	A1- 7	R04	4/76
3- 7	R04	4/76	4-35	R05	11/76	A1- 8	R04	4/76
3- 8	R04	4/76	4-46	R04	4/76	A1- 9	R04	4/76
3- 9	R04	4/76	4-37	R04	4/76	A1-10	R04	4/76
3-10	R04	4/76	4-38	R05	11/76	A1-11	R04	4/76
3-11	R04	4/76				A1-12	R04	4/76
3-12	R04	4/76	5- 1	R04	4/76	A1-13	R04	4/76
3-13	R04	4/76	5- 2	R04	4/76	A1-14	R04	4/76
3-14	R04	4/76	5- 3	R04	4/76			
3-15	R04	4/76	5- 4	R04	4/76	A2- 1	R04	4/76
3-16	R04	4/76	5- 5	R04	4/76	A2- 2	R04	4/76
3-17	R04	4/76	5- 6	R05	11/76	A2- 3	R04	4/76
3-18	R04	4/76	5- 7	R05	11/76	A2- 4	R04	4/76
			5- 8	R05	11/76	A2- 5	R04	4/76
4- 1	R04	4/76	5- 9	R04	4/76	A2- 6	R04	4/76
4- 2	R04	4/76	5-10	R04	4/76	A2- 7	R04	4/76
4- 3	R04	4/76	5-11	R05	11/76	A2- 8	R04	4/76
4- 4	R04	4/76	5-12	R05	11/76			
4- 5	R04	4/76	5-13	R05	11/76	A3- 1	R04	4/76
4- 6	R05	11/76	5-14	R04	4/76	A3- 2	R04	4/76
4- 7	R04	4/76	5-15	R04	4/76	A3- 3	R04	4/76
4- 8	R04	4/76	5-16	R05	11/76	A3- 4	R04	4/76
4- 9	R04	4/76	5-17	R05	11/76	A3- 5	R04	4/76
4-10	R04	4/76	5-18	R05	11/76	A3- 6	R04	4/76
4-11	R04	4/76	5-19	R05	11/76	A3-7/		
4-12	R04	4/76	5-20	R04	4/76	A3- 8	R04	4/76
4-13	R04	4/76	5-21	R04	4/76			
4-14	R04	4/76	5-22	R04	4/76	A4-1/		
4-15	R04	4/76	5-23	R05	11/76	A4-2	R05	11/76
4-16	R04	4/76	5-24	R05	11/76			
4-17	R04	4/76	5-25	R04	4/76	A5-1	R04	4/76
4-18	R04	4/76	5-26	R05	11/76	A5-2	R04	4/76

PAGE REVISION STATUS SHEET

PUBLICATION NUMBER 29-390

TITLE OS/32 MT PRM

REVISION R05 DATE November 1976

PAGE	REV.	DATE	PAGE	REV.	DATE	PAGE	REV.	DATE
A6-1	R05	11/76						
A6-2	R05	11/76						
A6-3	R05	11/76						
A6-4	R05	11/76						
I- 1	R04	4/76						
I- 2	R04	4/76						
I- 3	R05	11/76						
I- 4	R04	4/76						
I- 5	R04	4/76						
I- 6	R04	4/76						
I- 7	R04	4/76						
I- 8	R04	4/76						
I- 9	R04	4/76						
I-10/ I-11	R04	4/76						
I-12	R04	4/76						

PREFACE

This manual describes Revision 2 of OS/32 MT, the multi-tasking real-time operating system for the INTERDATA 32-Bit computer systems.

The major new features are:

Multiple task common segments with sizes limited only by the available memory.

Support of discontiguous memory.

Improved timer management, including repetitive interrupts and time slicing.

Double Precision arithmetic capability.

Support of new hardware, including 67MB disc and Mini I/O.

Better disc management, including directory pre-allocation and OS images in files.

Appendix 6 describes the new features in more detail.

This revision of OS/32 MT is upwards compatible with previous revisions; tasks which use task common or the reentrant library must be re-established with TET R02 to run under OS/32 MT R02.

TABLE OF CONTENTS

ILLUSTRATIONS

TABLES

CHAPTER 1
SYSTEM OVERVIEW

INTRODUCTION

OS/32 MT is a Multi-Tasking Operating System for the INTERDATA 32-Bit Architecture processors. Both background and foreground facilities are provided so that program preparation can proceed concurrently with real-time system operation. Built-in functions of OS/32 MT include system control via the operator's console, interrupt handling and I/O servicing. Data file management features are provided for any system equipped with direct-access storage media.

OS/32 MT is upwards compatible with existing 32-Bit Operating Systems. The minimum hardware requirements to support OS/32 MT are:

INTERDATA 32-bit processor with 96 KB of memory
Memory Access Controller (MAC)
Display Panel
Interval and Line Frequency Clock
Power Fail/Automatic Restart Option
Console Device

- Teletype on local TTY Interface
- CRT on local TTY or PASLA Interface
- Carousel on local TTY or PASLA Interface

Paper tape, magnetic tape, cassette or LSU
(required to boot in system - may be TTY or Carousel Paper Tape)
Disc or any 2 magnetic media

- 800 BPI 9-track magnetic tape
- 1600 BPI 9-track magnetic tape
- 2.5 MB disc
- 10 MB disc
- 40 MB disc
- 67 MB disc
- 256 MB disc

The reader should be familiar with the following documents describing the 32-Bit processors:

32-Bit Series Reference Manual, Publication Number 29-365
Model 7/32 Reference Manual, Publication Number 29-405
Model 8/32 Processor User's Manual, Publication Number 29-428

Other manuals related to OS/32 MT are:

OS/32 MT Pocket Guide, Publication Number 29-505
OS/32 Series General Purpose Driver Manual, Publication Number 29-384
OS/32 MT Program Configuration Manual, Publication Number 29-389
OS/32 MT Task Establisher (TET/32) User's Manual, Publication Number 29-412
ITAM/32 Reference Manual, Publication Number 29-541
OS/32 MT Program Logic Manual, Publication Number 29-391
OS/32 Disc Initializer Program Reference Manual, Publication Number 29-508
Common Assembler Language (CAL) User's Manual, Publication Number 29-375
Model 8/32 Micro-Instruction Reference Manual, Publication Number 29-438
Common Microcode Assembler Language (MICROCAL) User's Manual, Publication Number 29-478
Model 8/32 Writable Control Store (WCS) User's Guide, Publication Number 29-479

OS/32 MT protects the foreground environment from the effects of undebugged background tasks. Memory is protected via the Memory Access Controller (MAC), which also provides hardware relocation of both foreground and background tasks at run time.

OS/32 MT provides facilities for supporting up to 253 foreground tasks running concurrently. It supports up to 1 megabyte of memory, either as contiguous main memory or local memory and various shared memory banks.

OS/32 MT supports the single precision and double precision floating point features of the 32-Bit Processors. It also contains software floating point packages to emulate the hardware facilities, for those configurations without floating point support.

OS/32 MT provides the ability to generate up to 15 task common areas. It also contains support for the Precision Interval Clock (PIC) and Line Frequency Clock (LFC) to enable tasks to schedule task level interrupts of a periodic and non-periodic nature.

OS/32 MT contains file management techniques which include three types of file structures, and a disc directory structure such that each disc contains complete information concerning all its existing files. Both static and dynamic protection ITAM/32 provides telecommunications support for OS/32 MT. Refer to the *ITAM/32 Reference Manual* for a description of its facilities.

IMPORTANT FEATURES

Outstanding features of OS/32 MT are:

Multiple application programs can operate concurrently through the use of interleaving techniques.

Tasks are scheduled by priority, with 240 distinct priority levels available to the user. An optional time slice scheduler is provided which allows tasks of equal priority to share processor time.

A highly modular structure and a system generation utility program allows system elements to be easily added or deleted, thereby assuring the user of a compact, tailored OS environment.

The number of tasks in memory at any time may be as great as 255, limited only by the amount of memory available and by system generation considerations.

Tasks need not be totally memory-resident, but may be segmented and overlaid from any bulk storage device.

Calendar and time of day are maintained by the system; in addition, interval timing facilities are available to user tasks, with a resolution of one millisecond.

Tasks may request the activation and execution of other tasks, and may pass parameters to one another. Tasks may take traps on the reception of these parameters, thus providing timely response to communications between tasks.

Tasks may take traps upon the completion of a time interval or a series of time intervals, proceed I/O termination, or upon power restoration in case of power failure.

I/O operations are device-independent, allowing device re-assignment without having to alter existing software.

Comprehensive file management facilities are provided on direct-access devices; three distinct file structures are provided for safe and efficient use of the disc. I/O devices and disc files are referenced by name; files are allocated on a sector basis.

Sharable data and Reentrant Library facilities are provided.

A powerful command language (Command Substitution System) is provided which allows sequences of commands to be invoked by a single command.

SYSTEM COMPONENTS

The OS/32 MT system is divided into the following major groups:

System Manager (Command Processor)
Executive Functions
Task Manager
Timer Manager
Memory Manager
File Manager
I/O Subsystem
Resident Loader
Floating Point Support (optional)

System Manager

The System Manager handles all interactions between the system and the console device. It provides the operator interface to OS/32 MT. It executes as a task in OS/32 MT and is designed so that many functions are performed through Supervisor Calls. The System Manager contains routines to support the Command Substitution System (CSS), to do memory partitioning, and to support Direct Access devices. The System Manager controls all I/O requests to the Console and Log devices.

The Systems Manager accepts commands from the system console device, decodes them and calls the appropriate executor. It contains logic to provide the console operator with informative messages in case of error.

Command Substitution System (CSS)

The Command Substitution System routines provide the ability to build, execute and control files of OS/32 MT operator commands. CSS consists of routines to execute CSS operator commands, to manage the CSS buffers and to provide the command parameter substitution facility.

Direct Access Support

The System Manager provides the operator with the command functions necessary to allocate and delete files, display files, rewind, and backspace files assigned to user tasks. It also contains commands used when mounting and dismounting direct access volumes. These functions are executed via SVC 1 and SVC 7 calls.

Console Support

The System Manager controls user task communication with the system console. Because of this feature and the structure of Task Management, commands can be entered and executed while user tasks are active, even if tasks have assigned the console device.

Executive Functions

The Executive contains routines to handle supervisor calls. These include SVC 2, SVC 3, SVC 5, SVC 6, SVC 9, and SVC 14. All functions are performed on behalf of the calling task.

Task Manager

The Task Manager handles task scheduling functions. A task is controlled through a Task Control Block (TCB). At SYSGEN time the user determines the number of tasks the system being built is to contain. The user may choose to have up to 254 tasks (plus the system manager) in a system.

Timer Manager

OS/32 MT makes use of both a line frequency clock and a precision interval timer to provide user tasks with a flexible set of timer management/maintenance services. The following services are provided: time of day clock, day and year calendar, interval and time of day wait, interval and time of day trap, and driver time-out. Time Trap functions may be set up to occur periodically.

Memory Manager

The Memory Manager handles dynamic system space, partition memory space, and Task Common memory space. The Memory Management system is more fully described in Chapter 3.

File Manager

The File Manager contains the SVC 7 Handler and the intercept routines for SVC 1 requests to Contiguous, Chained, and Indexed Files.

I/O Subsystem

The I/O Subsystem is composed of the SVC 1 Handler, the Peripheral Device Drivers, and certain other routines, such as the System Queue Handler.

The I/O System consists of system routines and control blocks necessary to provide device independent I/O requests.

Resident Loader

The OS/32 MT resident loader loads tasks, overlays and library segments. The input to the resident loader must be created by the OS/32 MT Task Establisher (TET/32). TET/32 outputs 'load modules' which contain a Loader Information Block (LIB) followed by a memory image of the task or library. The LIB enables the loader to derive the various parameters of the load module.

OVERVIEW

Figure 1-1 shows the principal interactions between the major groupings of the Operating System, including foreground and background tasks, the Resident Library, and Task Common partitions. For clarity, many minor interactions between these module groupings are not shown. For a more detailed explanation of system interactions, the reader is referred to the *OS/32 MT Program Logic Manual*, Publication Number 29-391.

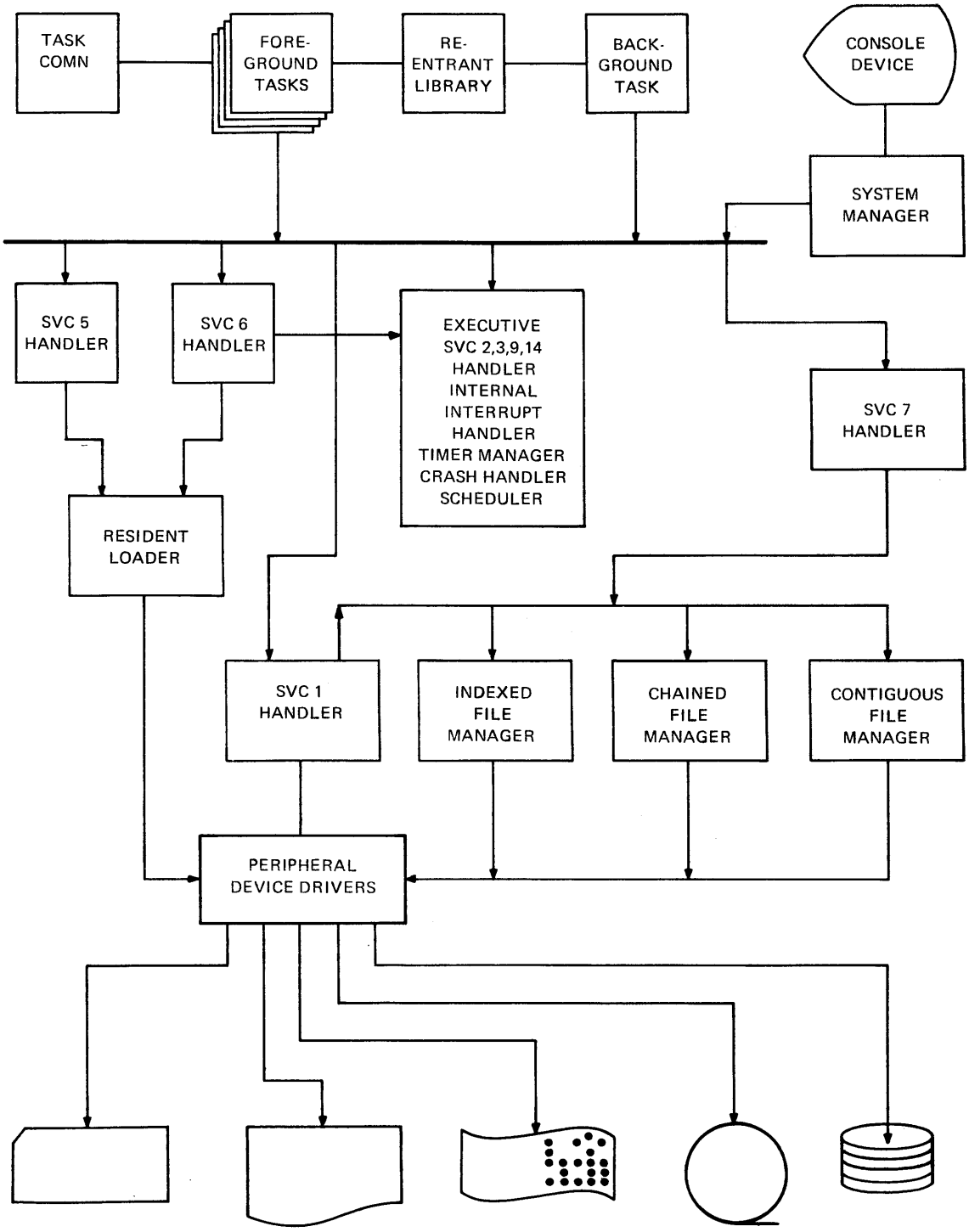


Figure 1-1. OS/32 MT Functional Block Diagram

HARDWARE CONSIDERATIONS

Floating Point Support

OS/32 MT provides an optional Floating-Point Emulation Package for those installations without Hardware Floating Point Support. At SYSGEN time, the user has the option of incorporating single precision, double precision or single and double precision floating point support into the target system.

In systems without hardware floating point, if no software floating point support is selected, no task in the system may execute floating point instructions. If a task does execute a floating point instruction, it will be treated as an illegal instruction. The Resident Loader does not load a task in a system without floating point support, if the LIB of its load module indicates that floating point is required.

If software floating point support is selected, a series of routines to simulate floating point instructions is included in the system.

If either software or hardware floating point is specified, and a task's options indicate it uses floating point, the operating system saves and restores the current contents of the task's floating point registers when the task is stopped and restarted. This means that each task has its own unique copy of the floating point registers.

Using Writable Control Store

The Model 8/32 Processor contains an optional area of high speed control store memory called Writable Control Store (WCS). The WCS provides the user with the ability to extend the architecture of the 8/32 Processor with user written microcode.

The INTERDATA Model 8/32 WCS Support Program (Program Number 03-102), provides the user with the ability to debug microcode routines written for WCS. This program executes as a task within an OS/32 MT Operating System environment.

When using WCS within OS/32 MT, the user should be aware that both E-tasks and U-tasks may use the instruction Enter Control Store (ECS), which causes control to be transferred to WCS. Transferring to Control Store may overwrite the OS image in memory, if:

1. WCS is not set up at all.
2. WCS, especially the first 16 words, is set up incorrectly.
3. WCS Support Program is at a lower priority than any other task in the system which uses WCS.
4. WCS image, maintained by the WCS Support Program, is destroyed by another E-task.

To avoid this, ensure the following:

1. WCS is set up all the time, i.e., the WCS Support Program must be at higher priority than any other task in the system which uses WCS.
2. WCS contains fully debugged microcode.
3. The first 16 WCS words are set up to point to the debugged microroutines or to the illegal instruction handler in the fixed ROM (ROM location X'208' labeled ILEGAL).

The instructions Branch to Control Store (BDCS), Read Control Store (RDCS), and Write Control (WDCS) are permitted only for use within E-tasks.

Improper use of these three instructions by an E-task can destroy the system. Hence, an E-task which uses these instructions, must be debugged beyond reasonable doubts.

Refer to the following manuals for detailed instructions concerning the use of WCS:

Model 8/32 Micro-Instruction Reference Manual, Publication Number 29-438.
Common Microcode Assembler Language (MICROCAL) User's Manual, Publication Number 29-478.
Model 8/32 Writable Control Store (WCS) User's Guide, Publication Number 29-479.

CHAPTER 2
SYSTEM OPERATION

SYSTEM GENERATION

OS/32 MT must be tailored to the specific configuration it is to support. This is accomplished by the OS/32 MT System Generation (SYSGEN) procedure. The OS/32 MT Configuration Utility Program (CUP/MT) allows the user to select the desired support from a library of object modules by specifying the desired target system with CUP/MT control statements. CUP/MT processes the input control statements and a library of peripheral device driver object modules to produce a series of programs. These are linked together with a library of system object modules by the OS/32 Library Loader (Program Number 03-065) to produce an absolute OS/32 MT load module. The load module can then be converted by TET/32 to an OS image, suitable for bootstrap loading.

An OS/32 STARTER system is supplied with OS/32 MT. OS/32 STARTER is a pre-SYSGENed OS/32 system which may be used to run CUP/MT.

SYSTEM START UP

OS/32 MT is initially loaded into the 32-Bit Processor by the 32-Bit Relocating Loader (Program Number 03-067) or the 32-Bit Direct Access Bootstrap Loader (Program Number 03-074). On completion of the load, control is transferred to OS/32 MT which prints:

```
OS32MTrr-uu
```

on the system console device, where rr is the release number and uu is the update number, and issues the command prompt:

```
*
```

OS/32 MT is then ready to accept commands.

Once OS/32 MT has been initially loaded, the *OS/32 Disc Initialize Utility Program* may be used to save an image of the OS on a disc. This disc image may be loaded by the *OS/32 Direct Access Bootstrap Loader Program.*

Once loaded, OS/32 MT provides both program control via Supervisor call (SVC) instructions and operator control via the System Manager command language.

ERROR HANDLING

There are three distinct types of error processing provided in OS/32 MT:

System Crash Handling
Unrecoverable Task Error Handling
Recoverable Error Handling

System Crashes

When OS/32 MT determines that further execution of the system may cause system or user data to be destroyed, the system crash handler is entered. A system crash code is displayed on the Console Display Panel and the system is put into a Wait state. Some of the conditions causing a system crash are:

Illegal Instruction in OS code
Invalid data on System Queue
Arithmetic Fault in OS code

For a complete list of system crash codes and their meanings refer to Appendix 5. After a system crash, *the system must be reloaded.*

Unrecoverable Task Errors

When OS/32 MT determines that further execution of a task may cause system or user data to be destroyed, the errant task is either paused with a message indicating the nature and location of the error or is abnormally terminated (ABTERMed) with a return code describing the error. Some of the conditions causing a task to be paused are:

Illegal Instruction in user task
Invalid address passed in SVC call
Attempt to access memory outside task partition

The user task may be continued, after correcting the error, or cancelled.

Recoverable Errors

Recoverable errors occur when OS/32 MT detects invalid data in a command, or insufficient data is supplied to perform the requested function. If the request is via a System Manager command, an appropriate error message is issued to the system console and the request is not performed. The operator may enter the correct command. If the request is via an SVC call, an appropriate status is returned in the PSW condition code or the SVC parameter block. The calling task may decide the proper recovery procedure.

SYSTEM SHUT DOWN AND RESTART

In a disc oriented OS/32 MT, it is necessary that the system be shut down or restarted in an orderly fashion to assure the integrity of the disc volumes in use. In particular, before shutting OS/32 MT down or restarting the operator should:

Cancel and delete (make non-resident) all tasks
Mark all disc devices off-line

OS/32 MT may now be restarted at location X'60'. If the system is restarted as a result of a system crash, the *Disc Integrity Check Utility* should be used to restore the integrity of the data on all disc volumes on-line at the time of the crash.

NOTE

If a system crashes, it should not be restarted at X'60'; it should be reloaded.

CHAPTER 3
SYSTEM DESCRIPTION

TASKS

The fundamental unit of work in OS/32 MT is the task. A task may consist of a single program, or it may include a main program and 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 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. Each task is controlled through a Task Control Block (TCB).

A task must be prepared by processing the component programs, subroutines and overlays with the OS/32 Task Establisher (TET/32). Once established, the task is loaded by the resident loader via the LOAD operator command or SVC 6. The following paragraphs describe task categorizations.

Foreground/Background Tasks

A task resident in a foreground partition (see the section entitled "Memory Management") is referred to as a foreground task while a task resident in the background partition is referred to as the background task. Foreground tasks have the full range of OS/32 MT services available; the background task has the following restrictions:

SVC 6 is treated as a NOP or Illegal SVC (according to the task options).
The Tasks's maximum priority is set at System Generation.
System Space limit is set at System Generation.
Background tasks may not communicate with the foreground tasks and vice versa.
Background tasks may not access Task Common Segments.

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

User/Executive Tasks

At task establishment time, a task may be designated a User task or an Executive task. User tasks (U-tasks) run in a protected mode (see the section entitled "Protection") while Executive tasks (E-tasks) have the full range of 32-Bit architecture capabilities available. E-task capabilities are designed to provide an orderly and well defined means for extending the system and are explained in the *OS/32-MT Program Logic Manual,* Publication Number 29-391.

Task Address Space and Logical Segments

The process of establishing a User task with the OS/32 Task Establisher (TET/32) produces a memory image load module of the task. The task references data and instructions via task space addresses, which are relative to the first location in the task, as if the task were loaded at location 0 in memory. When a task is loaded into the OS/32 MT system, the Memory Access Controller (MAC) is used to provide automatic relocation from task space addresses to real physical addresses, thus allowing a task to be loaded into any partition large enough.

Task address space is divided into one or more segments, where a segment is a set of contiguous program addresses starting on a 64K boundary. A maximum of sixteen segments (numbered 0 to 15) are available for each user task. The segment number is related to the starting address of the segment by the following formula:

SEGMENT NUMBER = START ADDRESS/64K

The following table shows the relationships of segment number to start address.

SEG NO.	Program Space START ADDRESS (Hex)
0	Y'00000'
1	Y'10000'
2	Y'20000'
3	Y'30000'
4	Y'40000'
5	Y'50000'
6	Y'60000'
7	Y'70000'
8	Y'80000'

SEG NO.	Program Space START ADDRESS (Hex)
9	Y'90000'
10	Y'A0000'
11	Y'B0000'
12	Y'C0000'
13	Y'D0000'
14	Y'E0000'
15	Y'F0000'

All segments are classified according to their contents, that is, either Impure code and data, Task Common or Reentrant Library. A user task may consist of the following program segments:

1. One main (impure) segment, or
2. A main segment and some combination of a Reentrant Library and one or more Task Common segments.

The impure segment consists of one or more contiguous physical segments beginning at segment 0 (address X'0').

A Task Common Segment consists of one or more contiguous segments whose address is specified at task establishment time.

The Reentrant Library segment is one segment and starts at location Y'F0000' (segment 15).

Refer to the *OS/32 MT Task Establisher (TET/32) User's Guide* for further information.

Status

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

Current
Ready
Wait
Paused
Dormant

Additionally, tasks are classified as either resident or non-resident. By definition, a task which is resident is not deleted when it completes execution. A non-resident task which goes to end-of-task (EOT) is deleted from the system. A task may be made resident at task establishment time or at run time by the operator or another task.

The Current Task is the task 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, and 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 dispatched (i.e., become Current) whenever it becomes the highest priority Ready task.

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

Wait State	Event
I/O wait	Waiting for I/O completion
Connection wait	Waiting for I/O to start
Time wait	Waiting for an interval or time of day
Trap wait	Waiting for a task-handled trap
Load wait	Waiting to be loaded
Task wait	Waiting to be released by another task

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 by another task. When a resident task goes to EOT it enters the Dormant state. When any task is loaded, it enters the Dormant state after load-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.

Priority and Scheduling

OS/32 MT recognizes 256 priority levels from a high of 0 to a low of 255. Of these levels, 10-249 are available to user tasks while 0-9 and 250-255 are reserved for the system's use. Each task has three priorities associated with it:

Maximum Priority
Task Priority
Dispatch Priority

Maximum priority is the highest priority that a task may be assigned; it is set at task establishment time. The maximum priority of a background task is set at SYSGEN time.

Task priority is the priority currently assigned to the task; it is initially set at task establishment time and may be modified by operator command or SVC 6.

Dispatch priority is the priority set up by the system to determine the order in which ready tasks are serviced. Normally, a task's dispatch priority is the same as its task priority but it may be raised temporarily if the task is using a system resource required by a higher priority task.

Two types of scheduling algorithm are available. Tasks may be scheduled in strict priority order or time-sliced within priority. In the former case, if two tasks of equal priority are started, a task remains active until it relinquishes control of the processor. Care should be taken in assigning priorities so that tasks which do not frequently relinquish control of the processor do not inadvertently lock out other tasks. A task may relinquish control in one of the following ways:

It is Paused by the console operator.
It is cancelled by the operator or another task.
A higher priority task becomes ready because of some external event.
It executes an SVC that places it in Wait, Paused or Dormant state.

Rather than scheduling on a strict priority basis, tasks may be time-sliced within priority. This option allows the user to ensure that tasks of equal priority receive equal shares of processor time.

The time-slicing option may be enabled and disabled by an operator command. Refer to Chapter 5 for further information.

When a task becomes ready, it is queued on a round-robin basis behind all ready tasks of equal priority.

Protection

User tasks run in a protected mode. They cannot access memory outside their boundaries, cannot execute code in Task Common, and cannot use any privileged instruction. Privileged instructions include all I/O instructions and any instruction that changes the state of the processor, such as LPSWR, EPSR.

In order to request I/O functions or any processor state change, user tasks must use the SVC instruction. (See Chapter 4.)

Memory protection is accomplished through the use of the MAC. This protection is transparent to user tasks running normally under OS/32 MT. Memory access errors by a task are handled either:

1. Automatically by the operating system, or
2. By the task itself in an error trap routine.

SYSTEM MANAGER

The console operator interface is provided by a task called the System Manager. The System Manager is an E-task which runs as the highest priority task in the system. The System Manager interprets and executes all commands; it also performs all I/O requests to the console device. Chapter 5 describes in detail the commands and procedures related to communication with the System Manager. The System Manager task is loaded as a part of OS/32 MT and cannot be cancelled. When the System Manager is the only task in the system, except for a dormant background task, the system is said to be quiescent.

SUPERVISOR CALLS (SVCs)

The program interface to the operating system is provided through Supervisor Call (SVC) instructions. SVC instructions are executed by programs to request OS/32 MT services. The parameters associated with the request are passed to the OS in a parameter block. Most of the services provided by the System Manager are performed with SVC instructions, thus making these services available to user tasks. Chapter 4 describes the individual SVC instructions and their associated parameter blocks in detail.

MEMORY MANAGEMENT

Memory in an OS/32 MT system is divided into two classes, local memory and global memory. Local memory is defined as that area containing the OS, all System Space, an optional Reentrant Library segment, an optional Task Common area (referred to as local Task Common), all foreground partitions, and the background partition. The total amount of local memory is known to the system as MTOP.

MTOP is defined at SYSGEN time.

NOTE

Local memory is always contiguous.

All Memory locations above MTOP are referred to as global or shared memory. Global memory may be physically contiguous to local memory, or may be located on shared memory banks. Using Multiport Memory, it is possible to configure 32-Bit Processors in systems with shared memory banks. In these multiprocessor systems, each system is able to address its own local memory and one or more shared memory banks. This memory need not be contiguous.

The current implementation supports up to 14 shared memory segments. The size, number and location of these segments are established at SYSGEN time. A new SYSGEN is required to vary any of these.

The configuration of shared memory in the OS is intended to correspond to the hardware configuration. The use of shared memory is restricted to Task Common segments.

Foreground Partitions

Up to 253 foreground partitions may be established at SYSGEN time. At this time, the initial size of each partition is set. When a task is loaded by the console operator or by an SVC 6 call from another foreground task, the task is loaded into a partition and the memory associated with that partition becomes the task's allocated memory.

Only one task can be loaded into any partition. To be loaded, a task must be able to fit into a vacant partition. The number of foreground tasks that may run concurrently is limited by the number of foreground partitions specified at SYSGEN time.

Normally, a task is loaded into the first partition (lowest memory address) large enough. However, a task may be loaded into a specific vacant partition by console operator command.

SVC 6 allows foreground tasks to request the loading and execution of other foreground tasks, to cancel these tasks or delete them from memory, or to communicate with these tasks. A foreground partition is referenced by the name of the task loaded in it, or from operator commands by its partition number if it is vacant. (See the section entitled "Sample Memory Allocation.")

Background Partition

OS/32 MT provides one background partition. This partition is restricted in the following manner: SVC 6 is treated as a NOP or an illegal SVC (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, although it may make use of the Resident Library.

The maximum priority and maximum amount of system space available to the background partition, are fixed at SYSGEN time. They cannot be adjusted by the operator. The size of the background partition is fixed and does not have variable bounds; it can be adjusted by the console operator when the size of foreground partitions are adjusted. The size of the background is determined by the amount of local memory not being used by other partitions. The background partition is always referenced by its name, .BG.

Task Common

Up to 15 task common areas are supported by OS/32 MT, one in local memory and up to 14 in global memory. Each area appears to the OS as a named Task Common segment. Task Common areas are sharable data segments and, as such, are in Execute Protected memory. Therefore the segments are writable and may contain data but may not contain executable code. Only foreground tasks may reference Task Common areas.

The console operator has control over local Task Common, but no control over global Task Common segments. Local Task common size is established at SYSGEN time; it is specified in multiples of 256 bytes. The size may be varied by the console operator, whenever the system is quiescent, with the SET PARTITION command.

Task Common areas in global memory are fixed at SYSGEN time. The number, size and starting addresses of all shared memory Task Common segments may be modified only by a new SYSGEN. Task Common segments may range in size from 256 bytes to the physical limit of the target configuration, in multiples of 256 bytes.

The sizes of Task Common segments made accessible to a task at load time are set by the corresponding common declarations in the source program, with the exception that a task which uses local Task Common only is given access to the whole of .TCM. This exception is provided for compatibility with previous releases.

Refer to the *OS/32 MT Task Establisher User's Manual* and the *OS/32 MT Program Configuration Manual* for information on how to establish and name Task Common segments.

Resident Library

One resident library partition is supported in OS/32 MT. This partition may be of any size from 256 bytes to 64 KB, in 256-byte increments; it is mapped into the using program's address space.

The size of the resident library partition is not established at SYSGEN time, but instead is established when the resident library is loaded. The library may be loaded by the console operator at any time when the system is quiescent; however, this is normally done only once, after system initialization.

Both foreground and background tasks may make use of the resident library. Calls to resident library routines are resolved when the task is established. See the *OS/32 Task Establisher User Manual* for details on establishing a task that uses a resident library. In operator commands, the resident library partition is referenced by its name, .LIB.

System Space

Certain memory areas in an OS/32 MT system are not occupied by any partition. These areas are known as system space. System space is used in OS/32 MT for two purposes: to hold the OS/32 MT 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. Dynamic data structures are located in the highest part of local physical memory. These are the only two areas of system space; the remainder of physical memory is devoted to partitions, task common, and the resident library.

At SYSGEN time the user is required to establish the area of memory for use by dynamic data structures. Currently, the following dynamic system data structures exist: File Control Blocks (FCBs) and Timer Queue Elements (TQEs). The minimum size of an FCB is 212 bytes (for a contiguous file). The maximum FCB size is a function of the block sizes chosen for Chained and Indexed files. The sizes are approximated by the following formulas:

Chained files: 256 bytes + 2* block size
Indexed files: 256 bytes + 2* data block size + index block size

where maximum block size is established at SYSGEN time. An FCB is created in system space each time a file is assigned to a Logical Unit. The FCB remains in memory as long as the file is assigned.

Each TQE is 24 bytes in size. TQEs are created for each time interval request.

The user should be aware of the size and frequency of system space requests when determining the size of the dynamic system space necessary.

Access to the dynamic data structure area is protected in OS/32 MT so that no user task can seize excessive space. The protection mechanism is as follows:

At task establishment time, a limit is set for each task to indicate the amount of system space it is permitted to request (via SVC calls). This limit should normally be set somewhat greater than the amount that the task is likely to require under normal circumstances. If that limit is exceeded through some program error, the system call that caused the attempted system space request is aborted.

In order to protect against an errant background task which may have been mis-established, the limit set at task establishment time is ignored for background tasks. Instead, a limit for background tasks is set at SYSGEN time with a CUP/MT control statement.

Sample Memory Allocation

Figure 3-1 shows a map of a hypothetical OS/32 MT system. The assumed configuration is a processor with 208 KB of local memory. The OS is presumed to occupy 64 KB. The structure is as follows:

Dynamic System Space:	24 KB
Local Task Common:	24 KB
Foreground:	3x8 KB, 16 KB, 24 KB
Resident Library:	8 KB
Background:	24 KB
Global Task Common at physical address X'34000'	16 KB
Global Task Common at physical address X'40000'	32 KB

The remaining 24 KB of local memory is system space, used for dynamic system data structures.

Notice that global task common number one ends at physical address X'38000', and that global task common number two begins at X'40000'. From X'38000' to X'40000' is a 32 KB gap in physical space. OS/32 MT supports such hardware configurations.

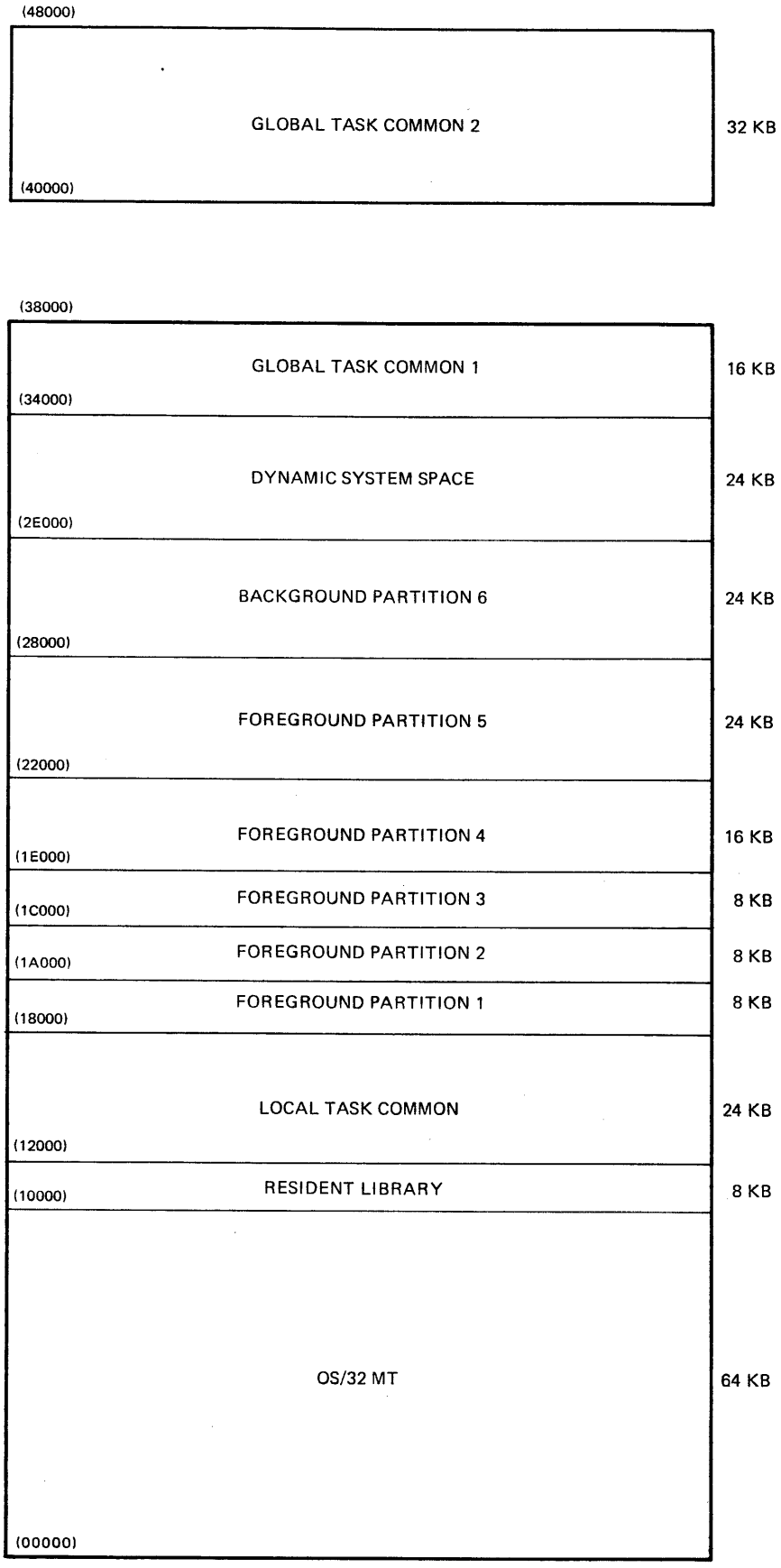


Figure 3-1. OS/32 MT Memory Map Example

INTERRUPTS AND TRAPS

All interrupts at the processor level, both external and internal, are handled by the operating system. OS/32 MT also 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 hardware and software-generated causes. A task handled trap may occur for the following reasons:

Power Restoration
SVC 14 Interrupt
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 another task
Illegal instruction
Memory Access Fault
Arithmetic Fault

Task Queue

Event-related information is maintained for each task within its own task queue. A task queue is a standard INTERDATA 32-Bit Series circular list. A task queue service trap occurs whenever that trap bit in the current TSW is set and when the task queue is non-empty.

Task Status Word (TSW)

Traps, and additions to the task queue, are controlled through the task status word (TSW), which is the task level analogue of a 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 (see Table 3-1). The initial TSW word of a task is set at task establishment time, default = 0. SVC 9 calls may be used to change the TSW when the task is running. At termination, the TSW is reset to zero. The TSW is set by initialization at TET time, SVC 9 calls, and TSW swaps on traps.

User Dedicated Locations (UDL)

The first 256 bytes of a task's address space are reserved for the User Dedicated Locations. The UDL contains TSW swap areas and other data used for communication between the operating system and the task. A map of these locations is shown in Figure 3-2. The UDL, with the appropriate areas defined, must be assembled as a separate program or as the first 256 bytes of one of the programs included in a task. The program containing the UDL must be biased at task space address X'00000' during task establishment. See the *OS/32 Task Establisher User's Manual,* Publication Number 29-412, for further information.

CTOP is a fullword. After an SVC 2 code 5 call CTOP contains the address of the highest halfword in the task's allocated memory (Impure Segment). This address does not reflect the possible use of the Resident Library or Task Common segments.

UTOP is a fullword. After an SVC 2 code 5 call UTOP contains the address of the first fullword following the user program. This value may be modified by the use of SVC 2 code 2 and code 3 calls. Its current value is obtained only after an SVC 2, 5 call. UTOP always contains an address aligned on a fullword boundary.

UBOT is a fullword. After an SVC 2 code 5 call UBOT contains the lowest address in the user program address space. For User tasks, this value is X'00000'.

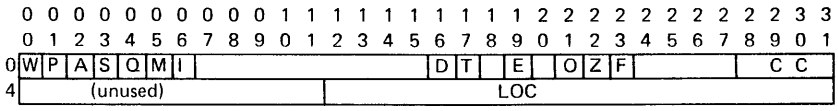
TSW swap areas are provided for Power Restoration, Arithmetic Fault, SVC 14, Task Queue Service, Memory Access Fault, and Illegal Instruction. These swap areas are each four fullwords in length. The first two fullwords of each location serve to save the previous TSW when a trap occurs; the new TSW is taken from the second two fullwords.

The field A (TASK QUEUE) contains the address of the task queue. This location must be set up by the program prior to enabling any entries to the task queue. If the content of this location is zero, no task queue entries are made, regardless of the state of the TSW enable bits for queue entry.

The field A (MESSAGE BUFFER) contains the address of a 76 byte storage area beginning on a fullword 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.

The location A (SVC14 ARG) is used by the operating system to save the effective address of an SVC 14 argument prior to performing an SVC 14 trap.

TABLE 3-1. TASK STATUS WORD



TASK STATUS WORD (2 FULLWORDS)

BIT	NAME	MASK	MEANING
0	W	Y'80000000'	Trap Wait: task is suspended until a trap occurs.
1	P	Y'40000000'	Power Restoration Trap Enable: trap is taken on restoration of power following any power failure.
2	A	Y'20000000'	Arithmetic Fault Trap Enable: trap is taken upon arithmetic fault.
3	S	Y'10000000'	SVC 14 Trap Enable: allows SVC 14 service. If this bit is not set, SVC 14 is illegal.
4	Q	Y'08000000'	Task Queue Service Trap Enable: any item added to the Task Queue when this bit is set causes a trap. Also, a trap is taken if a TSW having this bit set is loaded and the Task Queue is not empty.
5	M	Y'04000000'	Memory Access Fault Trap Enable: trap is taken when task attempts to address memory outside partition.
6	I	Y'02000000'	Illegal Instruction Trap Enable: trap is taken when task issues illegal instruction.
16	D	Y'00008000'	Enable Task Queue Entry on Device Interrupt.
17	T	Y'00004000'	Enable Task Queue Entry on Task Call: an SVC 6 Queue Parameter request directed at this task is rejected unless this bit is set.
19	E	Y'00001000'	Enable Queue Entry on Task Message: a message from another task can be received only if this bit is set; address of message buffer is added to queue.
21	O	Y'00000400'	Enable Queue Entry on I/O Completion: SVC 1 parameter block address is added to queue upon completion of I/O Proceed.
22	Z	Y'00000200'	Enable Task Queue Entry on Time-out Completion: the parameter in the SVC 2 code 23 (time trap) parameter block is added to task queue upon time-out completion.
23	F	Y'00000100'	Enable Queue Entry on SVC 15 Buffer Transfer Command Execution, Termination, or Halt I/O: SVC 15 is only supported by ITAM.
28-31	CC	Y'0000000F'	Current Condition Code, as in PSW.
			Bits 7-15, 18, 20 and 24-27 are reserved for future expansion, and should be set to 0. Bits 0-11 of the second word are unused and must be set to 0. Bits 12-31 of the second word contain the current LOC, as in the PSW.

0(0) CTOP	4(04) UTOP
8(08) UBOT	12(0C) RESERVED
16(10) A (TASK QUEUE)	20(14) RESERVED
24(18) A (MESSAGE BUFFER)	28(1C) A (SVC 14 ARGUMENT)
32(20) RESERVED	
48(30) POWER RESTORE OLD TSW SAVE AREA	
56(38) POWER RESTORE NEW TSW	
64(40) ARITHMETIC FAULT OLD TSW SAVE AREA	
72(48) ARITHMETIC FAULT NEW TSW	
80(50) RESERVED	
96(60) SVC 14 OLD TSW SAVE AREA	
104(68) SVC 14 NEW TSW	
112(70) TASK QUEUE SERVICE OLD TSW SAVE AREA	
120(78) TASK QUEUE SERVICE NEW TSW	
128(80) MEMORY ACCESS FAULT OLD TSW SAVE AREA	
136(88) MEMORY ACCESS FAULT NEW TSW	
144(90) ILLEGAL INSTRUCTION OLD TSW SAVE AREA	
152(98) ILLEGAL INSTRUCTION NEW TSW	
160(AO) RESERVED	
192(CO) USED BY SVC 6 "DELAY START" FUNCTION	

Figure 3-2. User Dedicated Locations

Task-Handled Traps

When a condition occurs that causes a trap, the current TSW (status and location) is saved in the appropriate area of the User Dedicated Locations (UDL). A new TSW (status location counter) 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. It is the trap routine's responsibility to save general and floating point registers as necessary prior to servicing the trap. An SVC 9 (Load TSW) is used to load the saved old 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. While in a Wait state, many such TSW swaps can occur, however, they are not detectable to the task until the wait condition is removed. At this time, the last TSW swap made determines where execution resumes. It is the responsibility of the user to assemble or dynamically prepare the desired new TSWs in the UDL for each type of trap.

Task Queue Service Traps

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

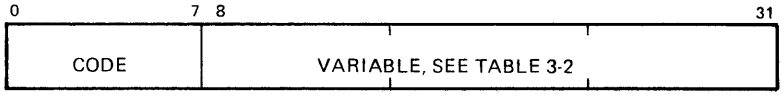
The following trap-causing conditions cause an item to be added to the task queue provided the appropriate bit is set in the TSW:

Queueing of a parameter to the task queue via SVC 6
Completion of an I/O proceed request
External Interrupt from a Trap Generating Device
Termination of a specified time delay
Message received from another task
SVC 15 Buffer transfer, Command execution, Termination, or Halt I/O

The queueing of each of these conditions is controlled by a TSW bit (as shown in Table 3-1). Each of these can be independently disabled, or enabled.

The task queue is a "circular list" in the standard INTERDATA list-processing format. Refer to the *Common Assembly Language (CAL) User's Manual* or appropriate processor reference manual for an explanation of this 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 it are rejected by SVC 6. A pointer to this list must be placed in the User Dedicated Locations by the program before trying to enable task queue entries.

The system always adds items to the bottom of the task queue. The format of the item added is as follows:



The CODE part of a task queue entry indicates the reason why the entry was placed on the queue. The content of the rest of the entry (bits 8-31) depends upon the reason code. Table 3-2 shows the reason codes and the content of bits 8-31.

NOTE

Reason codes not given in Table 3-2 are reserved for implementation in future releases.

Power Restoration Traps

A task may wish to be informed if system power has failed. Upon power failure, OS/32 MT 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 during a power fail is aborted. If the operation is on a direct-access device, the system retries it when power is restored. I/O operations to non-direct access devices return an error status to the task.

The normal course of events, in case of power failure, 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. Whenever there is a power failure, a TSW swap occurs upon restoration of power, and the task can go about the business of recovering.

TABLE 3-2. TASK QUEUE REASON CODES

CODE BITS 0-7	MEANING OF CODE	BITS 8-31
0	DEVICE INTERRUPT	PARAMETER ASSOCIATED WITH DEVICE
1	SVC 6 QUEUE PARAMETER	PARAMETER SPECIFIED IN CALL
6	MESSAGE RECEIVED	ADDRESS OF MESSAGE BUFFER
8	I/O PROCEED COMPLETE	ADDRESS OF SVC 1 PARAMETER BLOCK
9	TIMER TERMINATION	PARAMETER SPECIFIED IN CALL
10	SVC 15 BUFFER	ADDRESS OF SVC 15 PARAMETER BLOCK
11	SVC 15 COMMAND	ADDRESS OF SVC 15 PARAMETER BLOCK
12	SVC 15 TERMINATION	ADDRESS OF SVC 15 PARAMETER BLOCK
13	SVC 15 HALT I/O	ADDRESS OF SVC 15 PARAMETER BLOCK

SVC 14 Traps

The SVC 14 call is provided for the support of OS/32 AIDS (Program Number 03-064) the INTERDATA on-line debugging system. It may also be used by any task which is not currently using AIDS.

When a task executes an SVC 14, the S-Bit (bit 3) in its TSW should be set. If it is not set, the SVC 14 call is considered illegal. If the S-Bit is set, the effective address of the SVC 14 argument is placed in the User Dedicated Locations, and a TSW swap is taken.

Trap-Generating Devices

In certain applications, it is desirable for a task to be "awakened" in response to interrupts from some external device. OS/32 MT provides a set of facilities to do this.

Certain drivers, in particular the Eight-Line Interrupt Module driver, are capable of adding a parameter to a 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.

Currently, the only driver offered by INTERDATA that supports TGD functions is the Eight-Line Interrupt Module driver. Users may write their own TGD drivers; see the *OS/32 Series General-Purpose Driver Manual,* Publication Number 29-384, for details.

The functions provided by OS/32 MT for the handling of TGDs implement the entire ISA (Instrumentation Society of America) proposed standards for process control. These functions are:

Connect: attach a TGD to a task.
Thaw: enable interrupts on a TGD.
SINT: simulate an interrupt on a TGD (Addition to ISA standard).
Freeze: disable interrupts on a TGD.
Unconnect: detach a TGD from a task.

Interrupts

Internal Interrupts handled by the OS/32 MT Executive are: Arithmetic Fault, Illegal instruction, Machine Malfunction and Memory Access Fault. On Arithmetic Fault, Illegal Instruction, and Memory Access Fault a task may take a trap by setting the appropriate bit in its TSW.

Arithmetic Fault

When an Arithmetic Fault (AF) interrupt occurs in system code, the System Crash Handler is entered. If the fault occurs in user code, the following table shows the action taken, depending on the settings of the PSW Arithmetic Fault Interrupt bit, the TSW Arithmetic Fault Trap Enable bit and the Arithmetic Fault Pause/Continue task option.

PSW BIT (ON/OFF)	TSW BIT (ENABLE/DISABLE)	TASK OPTION (AFPAUSE/AFCONT)	ACTION OF TASK UPON ARITHMETIC FAULT
ON	e	p	PAUSED + MESSAGE
ON	e	c	TRAP
ON	d	p	PAUSED + MESSAGE
ON	d	c	IGNORED + MESSAGE
OFF	e	p	IGNORED
OFF	e	c	IGNORED
OFF	d	p	IGNORED
OFF	d	c	IGNORED

Illegal Instruction

If an illegal instruction is detected within system code, the System Crash Handler is entered. If the illegal instruction is detected within user code, and the Illegal Instruction Trap Enable bit is set in the task's TSW, a trap is taken. Otherwise the user task is Paused and a message is output to the system log.

This does not apply if the illegal instruction interrupt was caused by the execution of a floating-point instruction on a system using software floating-point traps. In that case, the floating-point traps routine is entered and the execution of the floating-point instruction is simulated.

Machine Malfunction

The machine malfunction interrupt occurs on memory parity error, power failure and power restoration, and when the INIT switch on the display panel is depressed.

A memory parity error causes a system crash if it occurs within system code; otherwise, the active task is paused and a message is output to the system console. Memory parity errors occur when addressing non-existent memory in a 7/32 Processor with the parity feature installed. Addressing non-existent memory in an 8/32 Processor within system code is an illegal operation, and the result is undefined.

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 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 whose TSW is set up to take a task-handled trap on power restoration. The SET TIME command should be used to set the clock as soon as possible following a power failure.

Memory Access Fault

This interrupt occurs when a task attempts to violate the conditions of memory protection imposed by the Memory Access Controller (MAC). The MAC is a hardware device contained within the 32-Bit series processors to monitor all memory accesses. For more information, refer to the *Model 7/32 Reference Manual,* Publication Number 29-428. If the Memory Access Fault Trap Enable bit is set in the task's TSW, a trap is taken. If the bit is not set, the task is paused and a message is output to the system log. If this fault should occur in system code, the System Crash Handler is entered.

FILES AND DEVICES

In order to provide device independent input and output, programs direct all I/O requests to a Logical Unit (LU) rather than a specific device or file. The system maintains a Logical Unit Table (LTAB) for each task. LU numbers, which range from zero to a SYSGENable limit (maximum 254), correspond to entries in the task's LTAB. The LUs referenced by a task must be assigned to specific devices or files by operator command or SVC 7 calls prior to their use. This allows different devices or files to be used without recompilation of the program. Devices may be marked off-line making them unavailable for assignment by user tasks.

Direct Access Files

All direct-access devices supported by OS/32 MT may be accessed through the OS/32 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. Each volume contains all the information necessary to process the data on that volume. When a direct-access device is marked off-line, it is referred to by the device mnemonic associated with the device at SYSGEN 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 Test and Formatter Program, 06-173, and initialized for OS/32 use by the OS/32 Disc Initializer, 03-081. The Disc Initializer can also be used to place an OS image suitable for boot loading on a direct-access volume.

Volume Organization

Allocation of space on an OS/32 volume is made in a flexible way, in order to reduce the adverse effects of possible defective sectors. Each sector occupies 256 bytes. Only one sector is specifically required to be valid; this is Sector 0, Cylinder 0, on which the system maintains the Volume Descriptor.

The Volume Descriptor has five fields:

Volume Name
Pointer to File Directory
Pointer to OS Image
Size of OS Image
Pointer to Allocation Map

The remainder of the Volume Descriptor is reserved for future expansion.

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

Pointer to OS Image and Size of OS Image fields exist for compatibility with previous operating systems.

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/32 Disc Initializer, 03-081. OS/32 MT does not modify any portion of the Volume Descriptor.

The Allocation Map is a bit-map containing one bit for each sector on the volume. Since a sector occupies 256 bytes, the bit-map overhead is 0.05% of the space on the volume. This map is used 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 linked list of one-sector blocks. A directory block contains up to five file entries. When a direct-access volume is initialized, the user has the option of allocating a 'fast access' directory. This type of directory has blocks that are chained together for optimal disc access time. See the *OS/32 Disc Initializer Manual,* Publication Number 29-508, for a complete description of this procedure. If the directory is not preallocated, new directory blocks are added as needed. New directory blocks are added to a 'fast access' directory, when the preallocated directory blocks are exhausted.

Identification of Files

An OS/32 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 the file resides.

The file name 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/32 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.
LIB	Reentrant Library image format.
OVY	Overlay image 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 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. At SYSGEN time each device in the system is assigned a device mnemonic of up to four-characters.

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/32 MT supports three file structures, Chained files, Indexed files and Contiguous files. Although these structures differ, in many cases the same data manipulations can be performed on all three. 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/32 MT file structures are each optimized for one specific form of access.

Chained Files

The Chained file is an open-ended file structure consisting of a chain of blocks. One fullword of each block is used by the system as a pointer (this pointer is not available to the user).

The pointer field of each block points to the next and previous block in the chain. By following the pointers, all the blocks in the file can be found, no matter where they are scattered on the volume. The pointer is bi-directional; that is, it can be used to follow the chain backwards as well as forwards. The chain is anchored at each end in the file directory. The chained structure of this file is completely transparent to the user.

When a Chained file is allocated, the physical block size of the file is specified in multiples of 256-byte sectors. The user's logical record size is independent of the physical block size. Blocking and deblocking of logical records is performed automatically by the system using two buffers in the File Control Block. The size of each buffer is equal to the physical block size of the file.

Because of its structure, the Chained file is optimized for sequential access. In order to proceed from any one block to any other, all intervening blocks must be read. This is the normal case in sequential access, but is time consuming in random access, unless the distances between successive random accesses are small.

The Proceed I/O trap facility is supported by Chained files. However, an I/O Proceed call to a Chained file is treated as a WAIT call, in that control does not return to the task until the I/O operation is complete.

All access privileges (see Section entitled "Dynamic Protection") may be requested for Chained files, however, shared write access is not permitted and if requested, the system automatically changes the request to exclusive write access.

Chained file I/O returns End of File (EOF) status under any of the following conditions:

- a read sequential operation is attempted at the end of the file;

- in random mode a Read or Write is attempted and the logical number specified is greater than the total number of logical records in the file (for a Read) or greater than the total number of logical records in the file plus one (for a Write).

End of Medium Status (EOM) is returned if a write operation is attempted and not enough space is available on the device containing the file.

If an I/O error occurs during the reading of a Chained file, the I/O is terminated and the I/O error status is returned to the user. If an I/O error occurs during the writing of a Chained file, data may have been lost. The system returns the file to its last known state, adjusts the file information in the File Control Block accordingly, and returns an I/O error status to the user. The user should then CHECKPOINT the file and issue a FETCH ATTRIBUTES call to obtain the current status of the file. See the section entitled "SVC7 - File Handling Services" for a description of the Checkpoint and Fetch Attribute calls.

A forward-file or backward-file operation positions a chained file at the end or beginning, respectively. If no filemark is found, an EOF error status results. EOF status is also returned if the user attempts to read or write beyond the end of the file's allocation.

Contiguous Files

The Contiguous file is a fixed-length file structure. All blocks of a Contiguous file are allocated contiguously on the volume. 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 write a Contiguous file in a random fashion.

Contiguous file I/O is non-buffered and transfers of variable amounts of data occur between the task's buffer, to the disc. 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 number of bytes specified have been transferred.

The Contiguous file supports a pseudo filemark capability that gives it some of the characteristics of a Magnetic Tape device. The pseudo-filemark is defined to be a 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. 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 following this filemark. The write-filemark operation results in writing X'1313' at the beginning of the current record.

A 'Test and Set' operation is currently implemented only on Contiguous files. It provides compatibility with previous INTERDATA Operating Systems. The Test and Set function allows a task to access a file or a sector within a file, and at the same time mark it as being in use. When a program issues a Test and Set, the system reads the data into the user's buffer, and before returning to the user, checks the first halfword of the data read in. If this halfword is zero, it forces it to X'FFFF' and rewrites the complete record back in its original location. It then sets the caller's Condition Code to zero. If the first halfword is non-zero, the caller's Condition Code is set to X'F', and the record is not modified. The SVC 1 function code for Test and Set is X'60'. The Test and Set operation requires Wait I/O (i.e., Bit 4 of the Function Code set) and it is assumed even if not set, although an I/O proceed queue item is added to the task queue, if the relevant bit is set in the TSW. When requesting a Test and Set operation the buffer specified in the SVC 1 Parameter Block should be a minimum of 1 sector in length. (See the section entitled "SVC 1 - Input/Output Request.")

Indexed Files

The indexed file is an open-ended file structure composed of two levels of physical blocks, a chain of index blocks and a series of data blocks. Each index block contains fullword pointers to one or more data blocks, depending on the number of blocks in the file. The index blocks are linked together; two fullwords in each index block are used as forward and backward pointers to form a doubly-linked list. The directory contains pointers to the first and last index blocks, but no pointers to data blocks.

The data block size, index block size, and logical record size are established by the user at allocate time and are fixed for the duration of the file. The data block size and index block size are specified in multiples of 256 bytes. As with the chained file structure, the logical record length is independent of the physical block size.

The indexed file is accessed with a buffered access method. The user program requests data transfer on a logical record basis. The actual I/O transfers are executed by the system on a block basis, using the system buffers located in the File Control Block (FCB). Blocking and de-blocking of logical records are performed by the SVC 1 intercept routines in the File Manager.

An indexed file may be accessed sequentially or randomly. Because of the physical structure of the file, random access is readily performed. For example, to read block 1 and then block 60, the indexed file structure requires an overhead read operation for the index block containing the pointers to blocks 1 and 60, whereas the chained file structure requires all blocks between 1 and 60 to be read.

The open ended structure of the indexed file allows the file to be extended in a sequential manner. This is done by writing a logical record numbered one greater than the number of existing records. If there are currently five records in a file, a request to write record six causes the file to be extended. However, if there are currently five records, a request to write record seven or higher causes an End of File status. The file may be up-dated by writing over an existing record.

As with Chained files, the Proceed I/O traps occur if requested, however, all proceed I/O calls are treated as wait calls.

Shared write access is not permitted on Indexed files. If shared write access is requested, the system changes the request to exclusive write access.

File Access Methods

OS/32 MT supports two methods of access to files: random and sequential. These methods may be intermixed without having to close and reopen 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. 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 previous pseudo file mark)

   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 next pseudo file mark)

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 to a Chained or 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 a Chained or 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 with EOF (End of File) status. 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, could 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 above.

File and Device Protection

Files and devices may be protected in two ways: statically and 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 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 System Manager.

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.
00	FF	....	Unprotected for write, unconditionally 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.

The protection keys of a device are defined at SYSGEN time and may be changed by the console operator only.

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

A file cannot be assigned with a requested access privilege if it is incompatible with some other existing assignment of that file. For example, a request to open 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 3-3 shows compatibilities and incompatibilities between access privileges.

TABLE 3-3. ACCESS PRIVILEGE COMPATIBILITY

	ERSW	ERO	SRO	SRW	SWO	EWO	SREW	ERW
ERSW	–	–	–	–	*	–	–	–
ERO	–	–	–	–	*	*	–	–
SRO	–	–	*	*	*	*	*	–
SRW	–	–	*	*	*	–	–	–
SWO	*	*	*	*	*	–	–	–
EWO	–	*	*	–	–	–	–	–
SREW	–	–	*	–	–	–	–	–
ERW	–	–	–	–	–	–	–	–

* = compatible; – = incompatible

Write Protection

All files on a disc volume may be protected from write operations by MARKing the disc on-line as a protected device (see the section entitled "MARK".) When a volume is write protected, only assigns for Shared Read Only (SRO) and Shared Read/Write (SRW) are accepted; SRW is changed to Shared Read Only. If the hardware write protected feature of a disc is enabled, the volume must be MARKed on as a protected volume. Refer to "Device and File Control Commands" for more information.

Null Device

If a task performs I/O requests via SVC 1 calls, but no actual transfer is desired (possibly during testing), the LU used in the transfer may be assigned to the null device. The OS/32 MT Configuration Utility Program configures every OS/32 MT 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. Write requests return with the buffer unchanged, and normal return status.

CHAPTER 4
SUPERVISOR CALLS

SVC INSTRUCTIONS

The SVC instruction enables the 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/32 MT. 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 fullword boundary.

NOTE

Reserved fields in SVC parameter blocks should be initialized to zero, to assure upwards compatibility with future releases.

Table 4-1 summarizes the SVCs supported by OS/32 MT.

TABLE 4-1. OS/32 MT SUPERVISOR CALLS

SVC		FUNCTION
1		GENERAL PURPOSE I/O OPERATIONS
2 code	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
	10	TIME WAIT
	11	INTERVAL WAIT
	15	PACK NUMERIC DATA
	16	PACK FILE DESCRIPTOR
	17	MNEMONIC TABLE SCAN
	18	MOVE ASCII CHARACTERS
	19	PEEK
	20	EXPAND ALLOCATION
	21	CONTRACT ALLOCATION
	23	TIMER MANAGEMENT
3		END OF TASK
5		FETCH OVERLAY
6		INTERTASK SERVICES
7		FILE MANAGEMENT
9		LOAD TSW
14		USER SVC
15		ITAM DEVICE DEPENDENT I/O

SVC Errors

OS/32 MT responds to syntax errors in an SVC instruction by issuing a message to the system console and pausing the task.

The two possible error messages and their causes are:

hh:mm:ss taskid: ILLEGAL SVC AT XXXXXX

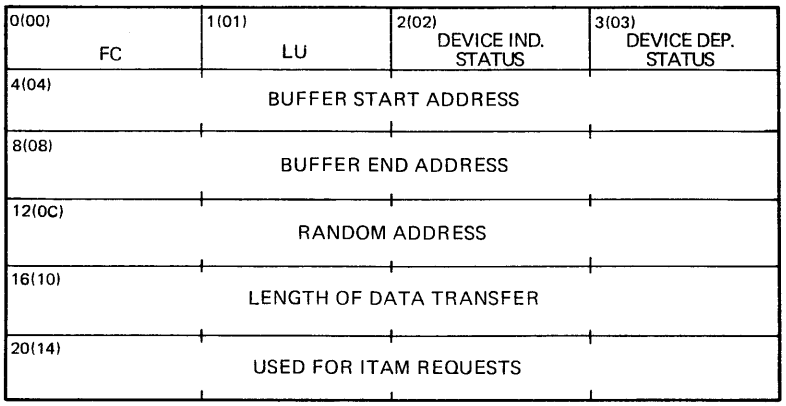
indicates an illegal SVC number, or invalid parameter specified in the parameter block.

hh:mm:ss taskid: INVALID ADDRESS IN SVC AT XXXXXX

indicates an invalid parameter block address (not in task's allocation; not on a fullword boundary) or an invalid address specified in a parameter block.

SVC 1 - INPUT/OUTPUT REQUEST

SVC 1 is used by a task to perform all general purpose I/O requests. The format of the SVC 1 parameter block is:



This parameter block must be on a fullword 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   4
PARBLK      DB      X'FC'       FUNCTION CODE
            DB      X'LU'       LOGICAL UNIT
            DS      2           STATUS
            DC      A(START)    START ADDRESS
            DC      A (END)     END ADDRESS
            DC      RANDOM      RANDOM ADDRESS
            DS      4           LENGTH OF DATA TRANSFER
            DS      4           USED FOR ITAM REQUESTS
```

Not all fields are required for every request.

Function Code

The Function Code field is used to specify Data Transfer requests and Command Function requests.

Data Transfer Requests

The function code field for each data transfer request is defined in Table 4-2.

TABLE 4-2. 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 00 - Wait only or Test I/O Complete
3	. . .x	ASCII/BINARY bit. This bit indicates the type of formatting requested. 0 - indicates ASCII formatting 1 - indicates binary formatting If bit 7 is set, this bit is ignored.
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 connection 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. **NOTE** If this is the only function code bit set, the request is interpreted as TEST I/O COMPLETE.
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).

In general, the request is defined by the logical 'OR' of the function code bits. The 0 setting of each bit is valid for all devices/files. If any invalid 1 setting of a bit is specified, the request is rejected 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 must be reserved for data transfer requests. Wait only and Test I/O complete requests (see the sections entitled "Proceed/Wait I/O" and "Wait Only") make use of the FC, LU, 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 command function requests is defined in TABLE 4-3.

TABLE 4-3. SVC 1 COMMAND FUNCTION CODE

BIT	ALIGNMENT	MEANING
0	x.	THIS BIT MUST BE SET TO INDICATE A COMMAND FUNCTION REQUEST
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

If bits 1-7 are all reset (function code X'80') the function is a Halt I/O Command.

The effect of all commands other than Halt I/O is device-dependent and is explained for each driver in the *OS/32 Series General Purpose Driver Manual,* Publication Number 29-384. The implementation of command functions for direct-access files is explained in the section of this manual entitled "File Access Methods."

SVC 1 Command requests are implicitly proceed calls.

When multiple bits in the FC byte are set, the following principle applies:

> The FC byte is scanned from left to right. The leftmost bit found that is meaningful to the device assigned to the specified LU indicates the function to be performed.

If no valid function is indicated by the FC byte, 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 by any Logical Unit by means of the Fetch Attributes SVC 7 call. (See the section entitled "SVC 7 File Handling Services.")

The full SVC 1 parameter block must be reserved for command function requests.

Halt I/O Command

A Halt I/O is used to cancel an I/O and proceed request which has previously been issued. This is especially useful on an interactive terminal device: if Halt I/O is not used, an outstanding read request must be satisfied before any other I/O can be started on a device.

The Halt I/O command is supported on the following devices:

> Card Reader, Printer, Paper Tape Reader/Punch, TTY Keyboard Printer, Magnetic Tape, TTY Reader Punch, Cassette, CRT, Carousel

Halt I/O is also supported on certain ITAM devices. Refer to the *ITAM/32 Reference Manual* for more information.

When a Halt I/O command is issued, the operating system schedules the I/O operation for termination. The actual termination is asynchronous to the Halt I/O request. When the I/O completes, the task receives a Proceed I/O Trap, if enabled. The parameter added to the task queue is the address of the original data transfer parameter block, not the address of the Halt I/O parameter block. Alternatively, the task may sense the completion of the I/O with SVC 1 Test I/O or Wait Only.

Two SVC 1 parameter blocks are involved in the Halt I/O processing. The first is the data transfer parameter block specified by the user when the I/O is initiated. The second is the command function parameter block which is requesting the Halt I/O. These should not be the same parameter block. As the result of Halt I/O, status is returned to both of these parameter blocks as indicated below:

1. Halt I/O parameter block status. The following status indications are returned to the Device Independent Status field of the user's parameter block as the result of requesting a Halt I/O:

Status	Description
X'00'	The requested I/O termination has been scheduled.
X'81'	LU not assigned.
X'82'	No I/O on-going for the task on this LU.

   The Device Dependent Status contains the device number.

2. Data Transfer parameter block status. When the I/O terminates, Status X'82' is returned to the Device Independent Status field of the data transfer parameter block. The Device Dependent Status field contains the information defined in the following section.

Logical Unit (LU)

In order to provide device independent I/O, all I/O requests are directed to a Logical Unit. LU is a number from 0 to a SYSGENed maximum (up to 254) which describes an entry in the task's LU table. The particular device or file desired must be assigned to the specified LU by operator command (see the section entitled "Task Related Commands") or SVC 7 call (see the section entitled "SVC 7 File Handling Services") 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 (see the section entitled "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 Condition Code is set to zero. In the case of a rejected Unconditional Proceed call, the Condition Code is set to X'F'. (See the section entitled "Unconditional Proceed.")

The Device Dependent Status byte may contain information unique to the specific type of device.

The system does not modify the contents of the Error Status halfword 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 are explained in the *OS/32 Series General Purpose Driver Manual,* Publication Number 29-384. The general definition of the status bits is given in Table 4-4.

TABLE 4-4. INTERPRETATION OF SVC 1 DEVICE INDEPENDENT STATUS BYTE

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'

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

For the 67 and 256 MB discs, if the device independent field contains an unrecoverable error status (X'84') the device dependent field contains the controller status. Refer to the Mass Storage Module Programming Manual 29-518 for a description of the possible values of this field. This information should not be interpreted at the programming level. It is provided as an aid in debugging possible hardware problems.

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 alone, 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 for a data transfer whenever the system cannot accept the SVC 1 FC byte as specified. This may be for one of the following reasons:

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

Buffer Address

For data transfer requests, the buffer is 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.

All buffers must start on a fullword boundary and must be fully contained in the same logical segment of task address space. Refer to section entitled "Tasks" for a description of program segments. Buffers used in read requests must be in a writable segment, since the memory locations are changed by the read operation.

An Invalid Address in SVC Error occurs if:

Buffer start and end are not in the same logical segment
Buffer end address is less than buffer start address
Buffer for a read request is in a write protected logical segment (e.g., .LIB)

Random Address

The Random Address field 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 driver dependent.

Length of Data Transfer

This field is used to return the actual number of bytes transferred during a request. This field is most useful when dealing with variable length record devices, such as Magnetic Tape. This field is undefined on error status.

Unconditional Proceed

Unconditional Proceed I/O is used when a task does not wish to wait for the requested operation. 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 suspended by the operating system until the device is free. At that time, the I/O request is initiated. If unconditional proceed is specified and the 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 use, the setting of the unconditional proceed has no effect on the request.

Proceed/Wait I/O

Once an I/O request has been initiated (that is, the specified device is free), any 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 data transfer. The status is not set until completion of the I/O (except for illegal function, and illegal LU which are rejected 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 (see the section entitled "Task-Handled Traps")
- Issuing a wait only request to the same LU to wait for completion

An I/O and Wait call (bit 4 of the function code set) requests the operating system to suspend the calling task until completion of the I/O operation.

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 outstanding I/O by the 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 tested is returned in the parameter block associated with the original I/O Proceed call and not in the Wait Only call's parameter block.

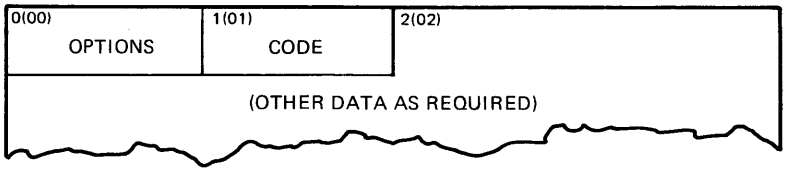
Test I/O Complete

A Test I/O complete call (function code X'02') returns with a condition code of X'0' if there is no outstanding I/O proceed to the specified LU by the task. If there is an outstanding I/O Proceed, the call returns a condition code of X'F'.

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, aligned on a fullword boundary. The general structure of an SVC 2 parameter block is as follows:



The CODE byte is a binary 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 are shown in Table 4-1.

The OPTIONS byte is generally used to further modify the conditions of the call. Options are 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 is as long as required for proper operation of the SVC 2 call.

SVC 2 Code 1 - Pause

This call places the task in Console Wait state. A message is issued to the system console. 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   4
PARBLK  DB      0,1     NO OPTIONS
```

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

SVC 2 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 task's memory allocation, but obtains locations from the program's current allocation. The parameter block format is as follows:

```
        ALIGN   4
PARBLK  DB      OPTION,2    DESIRED OPTION
        DC      H'REG'      ADDRESS REGISTER
        DC      F'SIZE'     NUMBER OF BYTES
```

Options allowed are:

X'00'	Get specified number of bytes
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 fullword boundary, requests are rounded up to the nearest fullword.

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 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 in REG. UTOP is not changed and the 'V' bit of the Condition Code is set (CC=4).

SVC 2 Code 3 - Release Storage

This call is the inverse of the GET STORAGE call (Code 2). It releases storage previously obtained. The format of the parameter block is as follows:

```
        ALIGN   4
PARBLK  DB      0,3             NO OPTIONS
        DCF     F'SIZE'         NUMBER OF BYTES TO BE RELEASED
```

This call does not reduce the task's memory allocation. The pointer UTOP is adjusted downwards by this call; but not below UBOT. Since UTOP always points to a fullword boundary, it is rounded up to the nearest fullword if the number of bytes released is not a multiple of 4 (e.g., a request to release 5 bytes actually releases 4). If the call adjusts 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.

Due to the alignment of the size field, there is an unused halfword in the parameter block.

SVC 2 Code 4 - Set Status

This call allows the user to modify the Arithmetic Fault (AF) Interrupt Enable bit (Bit 19) 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   4
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 whether or not the arithmetic faults are to be handled and the valid combinations are:

X'00'	Disable Arithmetic Faults; i.e., ignore Arithmetic Faults
X'10'	Enable Arithmetic Faults

See the section entitled "Arithmetic Faults," for an explanation of Arithmetic Fault Interrupts.

The CC byte parameter has the form:

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

SVC 2 Code 5 - Fetch Pointer

This SVC is used by a task that wishes to determine the extent of its memory allocation. It performs two functions:

– It returns a pointer to the base of the task's User Dedicated Locations (UDL).

– It copies UTOP, CTOP and UBOT from the task's TCB into its User Dedicated Locations. (See Figure 3-2.)

The first function is essentially meaningless for a U-task, since the User Dedicated Locations of such a task always begin at location 0 in the task's address space. An E-task, however, runs with the Memory Access Controller disabled. The address returned by this call, therefore, is the physical address of its UDL.

The second function is highly important. If the task has modified its UDL by writing into it, or if UTOP has been modified with Get/Release storage calls, the data contained in UTOP, CTOP and UBOT in the UDL may not be valid. This call restores these data to a valid state.

The format of the parameter block is as follows:

```
        ALIGN   4
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 start of the user's UDL is returned in this register.

SVC 2 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:

```
        ALIGN   4
PARBLK  DB      OPTION,6    DESIRED OPTION
        DCF     A(DEST)     ADDRESS OF DESTINATION
```

The OPTION field contains the supervisor call options. The ADDRESS OF DESTINATION field contains a pointer to the first location of 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 a 10 digit buffer is sufficient for unpacking a decimal number and an 8 digit buffer is sufficient for hexadecimal.

The converted number is right-justified in the buffer so that the least significant digit occupies the right-most 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 32-Bit constant.

Fullword alignment of the parameter block introduces a halfword of fill between the OPTION and A(DEST) fields.

SVC 2 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 on two forms, Direct and Indirect Text:

Direct Text:

```
        ALIGN   4
PARBLK  DB      OPTION,7    DESIRED OPTION
        DC      H'LENGTH'   LENGTH OF MESSAGE IN BYTES
        DC      C'TEXT'     TEXT OF MESSAGE
```

Indirect Test:

```
        ALIGN   4
PARBLK  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 expresses the length of the message in bytes. For the Indirect text option, this message may be of any length and may begin on any boundary.

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 attributes of the system console or log device. The System Manager, therefore, may restrict the Image option by executing it as though it were formatted for certain devices. The primary effect of the image option on a TTY 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. A second Log Message request causes a task to be placed in connection wait until its first request is complete.

SVC 2 Code 8 - Interrogate Clock

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 indicates 23:59:59. The clock may be interrogated in ASCII or binary format. The format of the parameter block is:

```
        ALIGN   4
PARBLK  DB      OPTION,8        DESIRED OPTION
        DCF     A(BUFFER)       ADDRESS OF BUFFER
```

Options recognized are:

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

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

hh:mm:ss

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

SVC 2 Code 9 - Fetch Date

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

```
        ALIGN   4
PARBLK  DB      0,9             NO OPTIONS
        DCF     A(DEST)         ADDRESS OF RECEIVING BUFFER
```

The receiving buffer must be eight bytes long and in a writable segment. The system returns the date in the following format:

mm/dd/yy

Alternatively, the following format may be selected at System Generation:

dd/mm/yy

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

SVC 2 Code 10 - Time-of-Day Wait

This call causes the calling task to be suspended until a specific time of day. The time of day is specified in seconds since midnight of the current day. Values greater than 86399 refer to future days. If the value given has already elapsed, the same time of the succeeding day is used. The format of the parameter block is as follows:

```
        ALIGN   4
        DB      0,10            NO OPTIONS
        DCF     Y'time'         TIME OF DAY (FULLWORD)
```

The time-of-day field is masked to a 28-Bit binary value.

This call requires that a timer queue element is obtained from system space. If the task has already exhausted its allocation, or if not enough memory is available, the call is rejected with the condition code = 4 ('V' bit set).

*Care should be taken when using this option since, depending on the console device, varying amounts of time must be left for a carriage return to take effect.

SVC 2 Code 11 - Interval Wait

This call causes the calling task to be placed in a Wait state for a given interval of time. The interval is defined in milliseconds from the time the call is executed. The parameter block format is as follows:

```
        ALIGN   4
        DB      0,11            NO OPTIONS
        DCF     Y'time'         INTERVAL IN MILLISECONDS (FULLWORD)
```

The Interval field is masked to a 28-Bit binary value. This call requires that a timer queue element be obtained from system space. If the task has already exhausted its allocation, or if not enough memory is available, the call is rejected with the condition code = 4 ('V' bit set).

SVC 2 Code 15 - Pack Numeric Data

This call is the inverse of the UNPACK BINARY NUMBER (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   4
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 User General Register Zero.

The pointer register is returned pointing to the first byte in the string that was not converted. (If decimal is specified, this would be the first non-numeric byte; if hexadecimal is specified, this is the first byte that is not 0-9 or A-F.) If register 0 is specified, 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, User General Register Zero is set to zero.

CC=4 ('V' bit set) means that the number processed was too large to be represented in 32 bits. If hexadecimal is specified, more than eight valid hexadecimal digits were processed; in this case, Register Zero contains the result of converting the least significant (right-most) eight characters. If decimal is specified, the number processed is greater than 2^{31}-1 (2,147,483,647). In this case, User General Register Zero contains the number processed, modulo 2^{31}.

SVC 2 Code 16 - Pack File Descriptor

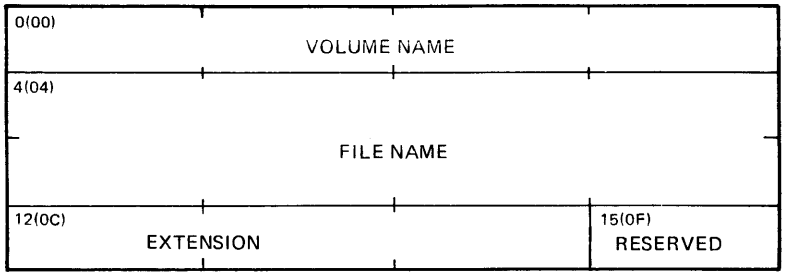
This call permits the user to process a File Descriptor in standard OS/32 Syntax. This call inserts the name of the system volume if the volume field is omitted. The format of the parameter block is:

```
        ALIGN   4
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 scan terminates when a syntax error is detected, or it proceeds until it has satisfactorily processed each field.

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



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 the "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 pointer contained in Register REG is returned pointing to the first byte that is not part of the file descriptor. The Condition Code is set on return as follows:

CC=	0	Normal
	1	no volume name present in input ('L' bit)
	4	syntax error ('V'bit)
	8	no extension present in input ('C' bit)
	9	no extension or volume name present in input ('C' & 'L' bits)

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 the "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:	DSC3ßßßßßßßßßßß	8 (no extension)
2.	ABC:FILE1.XY	ABCbFILE1ßßßXYßß	0 (normal)
3.	DEF:FILE2	DEFbFILE2ßßßßßßß	8 (no extension)
4.	FILE4.PDQ	****FILE4ßßßPDQß	1 (no volume)
5.	FILE5	****FILE5ßßßßßßß	9 (no volume or extension)
6.	FRED:FILE5.	FREDFILE5ßßßßßßß	0 (normal)
7.	$3%X,CP%	undefined	4 (syntax error)
8.	FILENAME 123	****FILENAMEßßßß	9 (no volume or extension)

**** = volume name field contents depend on selected option
b = blank character (X'20')

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.

In examples 1-6, REG is returned pointing to the first blank after the fd. In example 7 it is left unchanged.

In example 8, the receiving area contains the file descriptor FILENAME not FILENAME.123 since the scan terminates when it detects the blank (' ').

SVC 2 Code 17 - Scan Mnemonic Table

This call permits the user to decode command mnemonics in a way identical to the OS/32 MT Command Processor. The format of the parameter block is as follows:

```
        ALIGN   4
PARBLK  DB      0,17                NO OPTIONS
        DB      REG1,REG2           INPUT, INDEX REGISTERS
        DC      A(MNEMONIC TABLE)   MNEMONIC TABLE ADDRESS
```

The REG1 byte specifies the general register which points to the source string being scanned. The REG2 byte contains the number of a general register to contain the result. The MNEMONIC TABLE ADDRESS field contains the address of a mnemonic table within the user's program space. This table must begin on a fullword boundary.

A mnemonic table is composed of a string of mnemonics, 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	0 0	M N E M 2	0 0	M N E M 3	0 0 0 0

The first byte of a mnemonic may be any ASCII character; subsequent bytes of the mnemonic must be alphabetic. The scan terminates if it encounters a non-alphabetic character. A mnemonic may be of any length. Abbreviations are permitted in the same way as described in the operator command syntax (see the section entitled "Mnemonics"). 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, in which letters R, E, C are required, is coded as:

```
DB C'R'+X'80',C'E'+X'80', C'C'+X'80',C'ONFIGURE',X'00'
```

where the first three characters are flagged as required. Note the byte of zeros 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 to 1 (CC=4).

SVC 2 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:

```
        ALIGN   4
PARBLK  DB      OPTION,18           OPTION
        DB      REG1,REG2           INPUT, OUTPUT POINTERS
        DC      A(ECSTRING)         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 segment.

Options recognized are:

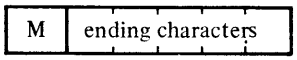
X'00' + N	No ending characters
X'80' + N	Use ending character string

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 for X'00', 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:



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. For example, a string containing the ending characters ,.;:/ is coded as follows:

```
DB    5, C',.;:/'
```

No alignment is required for the ending string.

SVC 2 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 28 bytes long. The format of this parameter block is illustrated in Figure 4-1.

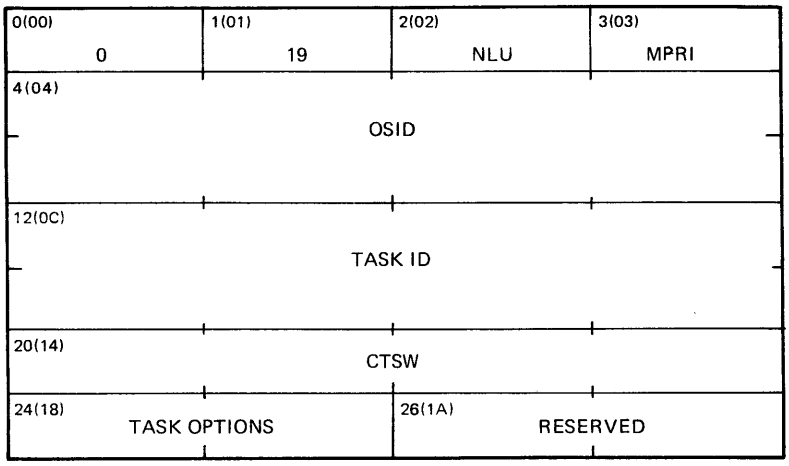


Figure 4-1. SVC 2 Code 19 Parameter Block

The first two bytes of the parameter block are set by the user program and indicate SVC 2 code 19. The remaining are return fields 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 "OS32MTrr" where "rr" is numeric and indicates the revision level of the OS/32 MT system.
TASK ID	Eight-character ASCII Task Identifier, left justified, blank filled as necessary.
CTSW	Status field (upper 32 bits) of the task's current Task Status Word.
TASK OPTIONS	The Options halfword from the task's TCB.

The parameter block must be on a fullword boundary in a writable segment and may be coded as follows:

```
        ALIGN   4
PARBLK  DB      0,19
        DS      26
```

NOTE

This parameter block must be 28 bytes long, otherwise, information following it is overwritten.

SVC 2 Code 20 - Expand Allocation

This SVC is provided for compatibility with previous 32-bit operating systems. The size of a task's allocation under OS/32 MT is defined by the size of its partition and is invariant while the task is in the system. The format of the parameter block is as follows:

```
        ALIGN   4
PARBLK  DB      OPTION,20       DESIRED OPTION
        DC      H'N'            NUMBER OF 256 BYTE BLOCKS
```

To allow programs written under previous operating systems to run unchanged, the following two option fields are recognized:

Option field = X'00'
Option field = X'80'

If option = X'00' is specified, the condition code 'V' bit is set (CC=4), indicating that the expand was not performed. If option = X'80' is specified, the condition code 'L' bit is set (CC=1), and N is set to zero, indicating that the expand was not performed. For option X'80' the parameter block must be in a writable segment.

SVC 2 Code 21 - Contract Allocation

This SVC is provided for compatibility with previous 32-bit operating systems. OS/32 MT takes no action upon receiving this call, and returns immediately to the caller. The parameter block may be coded:

```
        ALIGN   4
PARBLK  DB      0,21
        DC      H'N'
```

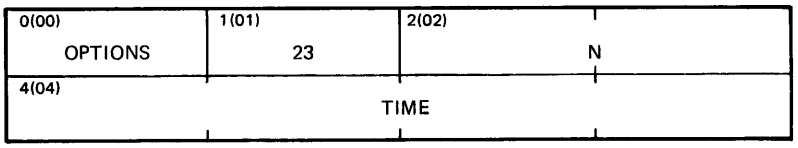
SVC 2 Code 23 - Timer Management

This call is used by a task to coordinate with real-time. A number of timer management facilities are available:

- A task can place itself in a time wait for a specified period.
- A task can ask to be trapped (using the task queue) at a specified time.
- A task can request repetitive traps (using the task queue) at defined intervals within a specified period.
- A task can determine the time until the occurrence of a specified trap.
- A task can cancel specified trap requests.

Parameter Block

The basic format of the SVC 2 code 23 parameter block is as follows:



which may be coded:

```
          ALIGN   4
PARBLOCK  DB      OPTIONS,23
          DC      H'N'
          DC      F'TIME'
```

The N and TIME fields take on different meanings for the specific options, and the parameter block is augmented for the repetitive traps request.

General Considerations

Some options require a timer queue element (TQE) to be allocated in system space. If the task has exhausted its allocation of system space, or if no system space is available, such calls are rejected with the'V'bit of the condition code set (CC = 4).

A task can make timer calls only on its own behalf and cannot affect or have any knowledge of outstanding timer requests for other tasks.

Timer requests can be intermixed without restriction, and intervals do not have to be requested in the order they expire. At EOT all outstanding intervals are cancelled.

Specific Options

The following table specifies the valid options to SVC 2 code 23. Any other values are illegal and result in the task being paused and an ILLEGAL SVC message to the system console.

BIT	HEX	BINARY	MEANING
0	X'00'	0000 0000	ADD TO QUEUE ON COMPLETION OF INTERVAL
0	X'80'	1000 0000	WAIT UNTIL COMPLETION OF INTERVAL
1	X'40'	0100 0000	ADD TO QUEUE REPETITIVELY
2	X'20'	0010 0000	READ TIME UNTIL SPECIFIED INTERVAL
3	X'10'	0001 0000	CANCEL INTERVAL REQUEST

Option X'00' Add to Queue on Completion of Interval

With this option a task requests that at a specified time in the future an item be added to its task queue. In the mean time, the task may continue processing or enter trap wait (using SVC 9) with the appropriate bits set in the TSW.

The form of this call is:

```
          SVC 2,TRAPTIME
          .
          .
          .
          ALIGN   4
TRAPTIME  DB      X'00',23      Option 00 , Code 23
          DC      H'N'          Specify register
          DC      F'TIME'       Specify time
```

where: N is in the range 0-15 and identifies the register, of which bits 8-31 contain the parameter part of the queue item to be added on termination of the interval (see Table 3-2).

TIME is a fullword quantity which specifies the interval. Bits 0-3 of the time field specify the type of interval required. The defined interval types are:

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

All other values are illegal and result in the task being paused and an ILLEGAL SVC message to the system console. The remaining 28 bits of the time field specify the duration of the interval. For time of day intervals values in excess of 86399 refer to future days.

Error Conditions

If a TQE is unobtainable, the call is rejected with the 'V'bit of the condition code set (CC = 4).

If, on expiration of the time interval, the Z Bit of the TSW (see Table 3-1) is not set, or A(Task Queue) in the UDL (see Figure 3-2) is zero, no item is added to the queue and the interval request is deleted (effectively the task has lost an interrupt).

If, on expiration of the time interval, the attempt to add the item to the task queue results in queue overflow, the task is abnormally terminated with a return code of 1000.

Option X'80' Wait Until Completion of Interval

With this option a task places itself in Time Wait for a specified interval.

The form of this call is:

```
                SVC       2,WAITTIME
                .
                .
                .
                ALIGN     4
WAITTIME        DB        X'80',23        Option '80', Code 23
                DS        2               N field not used
                DC        F'TIME'         Specify time
```

where: N is not used for this option.

TIME is a fullword quantity which specifies the interval. The interpretation of the time field is the same as for option X'00'.

Error Conditions

If a TQE is unobtainable, the call is rejected with the'V'bit of the condition code set (CC = 4).

For a time of day wait, if the time specified has already passed the task waits until the same time on the next day.

Option X'40' Add to Queue Repetitively

With this option a task requests that items be added to its task queue repetitively at defined intervals within a specified period. All the intervals in a period are of the same type. The periodic interrupts repeat until the task cancels the period or goes to EOT.

The form of this call is:

```
                SVC       2,PERIODIC
                .
                .
                .
                ALIGN     4
PERIODIC        DB        X'40',23              Option '40', Code 23
                DC        H'N'                  N items in table
                DCF       Y'T0000000'+A(TIMTABLE)  Interval type and pointer to table
                .
                .
                .
                ALIGN     4
TIMTABLE        DC        F'TIME'               1st Interval
                DC        F'PARAMETER'          1st Parameter
                DC        F'TIME'               2nd Interval
                DC        F'PARAMETER'          2nd Parameter
                .
                .
                .
                DC        F'TIME'               Nth Interval
                DC        F'PARAMETER'          Nth Parameter
```

where: N defines the number of intervals in the period; N can take any value.

T defines the interval type, exactly as for bits 0-3 of the TIME field in option X'00'.

A(TIMTABLE) is a pointer to N pairs of fullword items, defining the N intervals and the task queue items associated with each. Within each pair of fullwords, bits 4-31 of the first define an interval and bits 8-31 of the second specify the task queue item parameter (see Table 3-2).

Time of Day Intervals

If the type of interval is time of day (T=0) each time value represents the number of seconds from midnight of the day on which the call is made. Each time value must be at least one greater than the previous entry, i.e., the table must be arranged in increasing order. Values greater than 86,399 refer to suceeding days, thus 86,400 refers to midnight on the second day. The period is defined as time of the last interval, rounded up to the next multiple of 86,400, i.e., the period is always a whole number of days.

Elapsed Time Intervals

For elapsed time intervals (T=1) successive times in the table are incremental, each value specifying milliseconds from the previous value. The first value specifies milliseconds from zero, i.e., the time of the execution of the SVC 2 code 23. A time interval of zero is invalid in a repetitive request.

The period is defined as the sum of all the intervals in the table.

Parameters

Each interval in a periodic table can have a unique parameter associated with it. The parameters are returned to the task in the queue items added to the task queue when intervals expire. Parameters are limited to 24 bits (see Table 3-2).

Error Conditions

If a TQE is unobtainable, the call is rejected with the'V'bit of the condition code set (CC = 4).

If a time of day periodic table is out of order, or if an elapsed time periodic table has a zero interval, the task is paused and an ILLEGAL SVC message is issued to the system console.

If, on expiration of any time interval, the Z Bit of the TSW (see Table 3-1) is not set, or A(Task Queue) in the UDL (see Figure 3-2) is zero, no item is added to the queue (effectively the task has 'lost' an interrupt). The repetitive request is *not* cancelled.

If, on expiration of any time interval, the attempt to add the item to the task queue results in queue overflow, the task is abnormally terminated with a return code of 1000.

Option X'20' Read Time Until Specified Interval

This option is used by a task to determine the time until the expiration of an interval, previously established with either an option X'00' or option X'40' call. The call must specify both the parameter associated with the interval and its type.

The form of the call is:

```
                SVC     2,GETTIME
                .
                .
                ALIGN   4
GETTIME         DB      X'20 ,23'       Option '20', code 23
                DC      H'N'            Specify register
RTNTIME         DC      Y'T0000000'     Time type; used for return value
```

where: N is in the range 0-15 and identifies the register, of which bits 8-31 contain the parameter of the interval of interest.

T identifies the interval type (elapsed time, or time of day) of the interval of interest.

the fullword at RTNTIME is used to return the time until the specified interval. Therefore the parameter block for this option must be in a writable segment.

Effects

On return from this call the TIME field contains the time until the next expiration of an interval with the specified parameter. The value returned depends on the interval type. For time of day intervals (T=0) the value is the time in seconds from midnight on the day of the call. For elapsed time intervals (T=1) the value is the number of milliseconds remaining.

WARNING

IF MULTIPLE INTERVALS HAVE THE SAME PARAMETER, THE VALUE RETURNED IS THE TIME OF THE FIRST FOUND, NOT NECESSARILY THE NEXT INTERVAL TO EXPIRE.

NOTE

The T field is not changed by this call, thus, for elapsed time intervals, the value returned is Y'T0000000' + TIME.

If no interval is active with the specified parameter the'V'bit of the condition code is set (CC = 4).

Option X'10' Cancel Interval Request

This option is used to cancel previous requests for interval traps or for periods of interval traps.

The form of the call is:

```
                SVC     2,CANCELTM
                .
                .
                .
                ALIGN   4
CANCELTM        DB      X'10',23        Option'10', Code 23
                DC      H'N'            Specify register
                DC      Y'T0000000'     Time type
```

where: bits 8-31 of the register identified by N specify a parameter associated either with a single interval, previously established with an option X'00' call, or with any interval in a period, previously established with an option X'40' call.

T specifies the interval type of the intervals or periods to be cancelled.

Effects

All previous interval requests of the interval type specified, associated with the specified parameter, are cancelled. If any of these intervals is a member of a repetitive period, the whole period is cancelled.

If no interval is found of the right type with the specified parameter, 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, truncated to 8 bits. It replaces the Return Code used by the Command Substitution System (CSS) (see Chapter 5).

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.

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 non-resident foreground task, its files and devices are all closed. It is removed from memory by deleting all control information pertaining to the task. If, however, the task issuing this call is a resident foreground task, or is the background task, its files are checkpointed but not closed, and it is not removed from memory. See the section entitled "File Handling Services" for an explanation of file checkpoint and close operations.

SVC 5 – FETCH OVERLAY

This call permits a task to fetch an overlay from a specified Logical Unit. The SVC 5 parameter block is 12 bytes long. Its format is:

```
                ALIGN   4
PARBLK          DC      C'OVLYNAME'     OVERLAY NAME
                DS      1
                DB      OPTIONS         OPTIONS
                DC      H'LU'           LU NUMBER
```

Options recongized 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
X'20'	Mismatch on overlay name
X'40'	Overlay would not fit in allocated memory

If the overlay name is less than 8 characters, the field must be left justified and padded with blanks.

The eight-character overlay name field is matched against the overlay name in the Loader Information Block of the overlay file. If it does not match, error status X'20' is returned. The overlay file must be assigned to the Logical Unit specified in the parameter block. This file must be prepared by the OS/32 MT Task Establisher (TET/32). See the *OS/32 Task Establisher User's Manual* for details on preparing tasks with overlays.

The calling program is placed in load wait until the overlay is loaded. If the overlay is successfully loaded, the root program may Branch and Link to it as a subroutine.

Overlays should not call other overlays directly; the calling code is overlaid with the new overlay and the results are unpredictable.

The parameter block must be in a writable segment.

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

SVC 6 – INTERTASK COORDINATION

SVC 6 provides facilities for foreground tasks to invoke and communicate with other foreground tasks. SVC 6 is treated as an illegal SVC or a NOP from a background task, depending on the task options 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
Use Trap Generating Devices
Set task resident, non-resident
Suspend a task
Release a suspended task
Send a message to another task

The SVC 6 parameter block is 48 bytes long and must start on a fullword boundary in a writable segment. The format is shown in Figure 4-2.

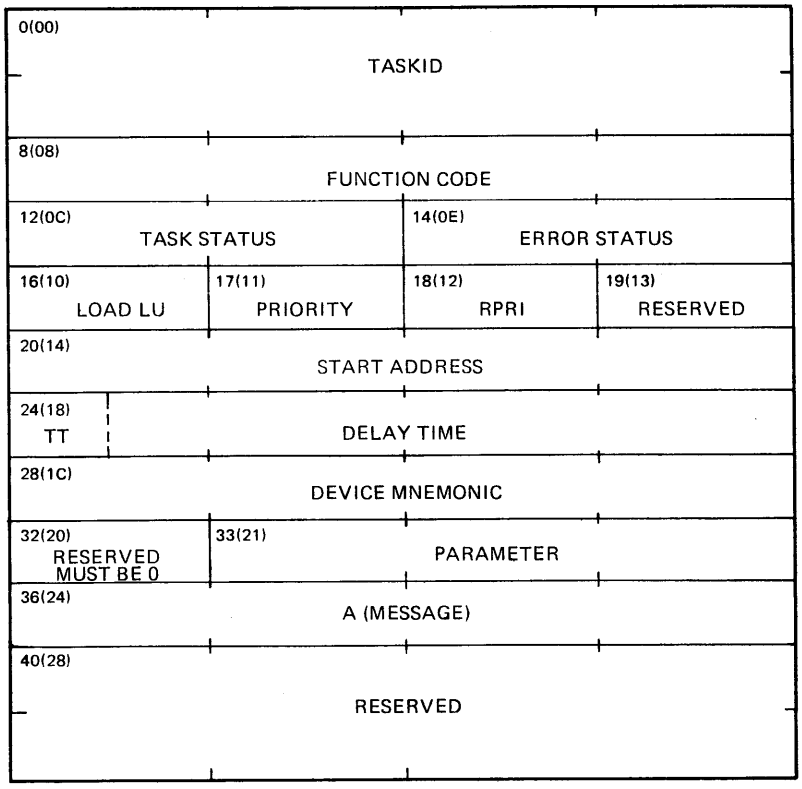


Figure 4-2. SVC 6 Parameter Block

The meaning of each field is explained in the description of each function requiring that field. Table 4-5 summarizes the use of each field. The parameter block may be coded:

```
        ALIGN   4
PARBLK  DC      C'TASK ID'      TASK ID (8 BYTES)
        DC      Y'function'     FUNCTION CODE (4BYTES)
        DS      4               STATUS. FIELD ($ BYTES)
        DB      LU              LOAD LU (1 BYTE)
        DB      PRI             PRIORITY (1 BYTE)
        DS      2               (2 BYTES)
        DC      A(START)        START ADDRESS (4 BYTES)
        DC      Y'TIME'         TIME DELAY (4 BYTES)
        DC      C'DEVM'         DEVICE MNEMONIC (4 BYTES)
        DC      Y'PARM'         PARAMETER (4 BYTES)
        DC      Y'MESS;'        MESSAGE BUFFER ADDRESS (4 BYTES)
        DS      8               (8 BYTES)
```

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

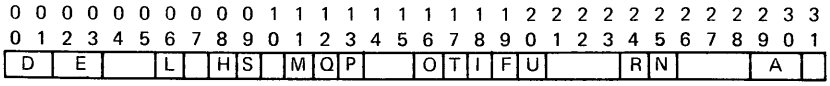
TABLE 4-5. SVC 6 PARAMETER BLOCK FIELDS

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 4-6.
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 4-8.
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 for E-tasks. Must be in range 10-249 for U-Tasks.
18	RPRI	Set by system to actual priority of specified task
19		(reserved)
20-23	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/32 time.
24, 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
24, Bits 4-31	Delay Time	Specifies number of second or milliseconds, as defined by TT field, to delay the start operation. Used only for Delay-Start.
28-31	Device Mnemonic	Specifies device affected by Connect, Thaw, SINT, Freeze and Unconnect functions. See Table 4-6.
32		Unused, must be zero.
33-35	Parameter	Specifies parameter to be queued for Queue parameter function; specifies device parameter for Connect function.
36-39	A (Message)	Specifies address of message Buffer for Send Message function.
40-47		(Reserved)

Task ID and Function Code

The Task ID field specifies the name of the task at which the SVC 6 is directed. A TASKID must consist of alphanumeric characters with the first character alphabetic. If the call is directed at the calling task (function code D field) then it is termed a self-directed call and the TASKID field is not required. A function may be directed at the calling task by specifying "other task" in the function code (see function code D field) and the calling task's name in the TASKID field. The function code specifies the functions to be performed on the specified task. Each bit in the function code 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 4-6.

TABLE 4-6. SVC 6 FUNCTION CODES



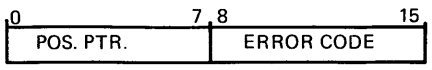
FUNCTION CODE FULLWORD

BIT(S)	NAME	HEX MASK	MEANING
0-1	D		Direction: 00,01 = illegal codes
		8000 0000	10 = other task
		C000 0000	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			Reserved
6	L	0200 0000	Load Task
7			Reserved
8	H	0080 0000	Task resident
9	S	0040 0000	Suspend execution
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.
21-23			Reserved
24	R	0000 0080	Release suspended task
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

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 indicates 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.

Error Codes are described in Table 4-7. The format of the Error Status halfword is as follows:



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 (0:31).

Status Return

No matter what functions are specified or whether or not an error is detected, the Wait Status halfword and the current priority from the specified task's TCB are always returned in the TASK STATUS and RPRI fields at the time of the SVC 6 call. 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 4-7. SVC 6 ERROR CODES

CODE HEX (DEC)	FUNCTION	MEANING
0(0)	All	No errors; all requested functions complete
1(1)	All	Syntax error in TASKID 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	Task requires floating point facilities not supported by SYSGEN
7(7)	A	Specified task not dormant
A(10)	A (delay)	Invalid code in TT field
B(11)	M	Message not sent
C(12)	Q	No queue, full queue, or entries disabled
D(13)	O,T,I,F,U	No such device in system
E(14)	O,T,I,F,U	Device named is not a connectable device
F(15)	O	Device is busy, cannot connect
10(16)	T,I,F,U	Device not connected to specified task
11(17)	L	Invalid or unassigned Load LU
19(25)	I	Device is not SINTable
41(65)	L	No partition with sufficient number of LUs
42 (66)	L	RTL or TCOM not present
44(68)	L	Invalid format on Loader Information Block
49(73)	L	No vacant partition of correct size
80-FF (128-255)	L	I/O error reading Load LU; error status is as returned by SVC 1

End Task Function (Function Code E field)

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 non-resident 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 non-resident foreground task. If this call is self-directed, SVC 6 processing is terminated.

Load Task Function (Function Code L field)

TASKID, 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 TASKID or the TASKID is invalid, the call is rejected. A self-directed load request is also rejected.

The specified LU must be assigned and positioned to the Loader Information Block (LIB). A foreground partition is then found which is vacant, large enough, and has sufficient Logical Units to accommodate the task to be loaded. If there is no such partition, the call is rejected. Upon loading the task, it is given the name found in the TASKID field. This call may not be used to load a task into the background partition; tasks may be loaded into the background partition only by the console operator.

On return from this call, the specified file or device is positioned after the loaded task. If the same task is to be loaded into more than one partition, the logical unit must be rewound prior to each subsequent load.

Task Resident (Function Code H field)

TASKID and Function Code are the required fields.

This function sets the called task memory resident.

Suspend (Function Code S field)

TASKID and Function Code are the required fields.

This function causes the called task to enter a Wait state. The task does not continue to execute until it is released by an SVC 6 Release call. A task may suspend itself. In that case, another task must be available to release it subsequently.

Send Message (Function Code M field)

TASKID and Function Code are the required fields.

This function provides the calling task with the capability of sending a message to the called task. This message can be up to 64 bytes in length. In the parameter block, the A(Message) field points to a 64 byte message buffer. The message is transmitted to the called task in binary image. No formatting of the message text is performed, although the system puts a header on the message, which consists of the 8-byte TASKID of the task that sent the message. The called task must be set up to receive messages. Refer to the section further in this chapter entitled, "Message Rings and Message Buffer Structures" for more information.

Queue Parameter Function (Function Code Q field)

TASKID, 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 additions to the queue. Otherwise, the call is rejected with appropriate error status.

Change Priority Function (Function Code P field)

TASKID, Function Code and Priority are the required fields.

This function changes the priority of the specified task to that specified in the PRIORITY field of the parameter block and sets RPRI to the old priority of the task. The call is rejected 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, the established maximum priority is used instead of the specified priority. This is not considered an error.

Trap-Generating Device Functions

The functions Connect, Thaw, SINT, Freeze and Unconnect are provided for the control of Trap-Generating Device (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 the TGD places in the Task Queue can be used to differentiate between multiple TGDs connected to a task.

Connect (Function Code O field)

TASKID, 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 (Function Code T field)

TASKID, Function Code and Device Mnemonic are the required fields.

This function enables interrupts on the named 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 (Function Code I field)

TASKID, Function Code and Device Mnemonic are the required fields.

This function simulates an interrupt on the named device. 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 the *OS/32 Series General Purpose Driver Manual,* Publication Number 29-384.) 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 (Function Code F field)

TASKID, Function Code and Device mnemonic are the required fields.

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

Unconnect (Function Code U field)

TASKID, Function Code and Device Mnemonic are the required fields.

This function detaches the named 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.

Release (Function Code R field)

TASKID and Function Code are the required fields.

This function takes an SVC 6 suspended task out of its Wait state. The task continues to execute as before it was suspended, provided it is not in any other Wait state.

Task Non-Resident (Function Code N field)

TASKID, and Function Code are the required fields.

This function sets the called task non-resident.

Start Task (Function Code A field)

TASKID, Function Code and Start Address are the required fields.

TT and Delay Time may optionally be specified.

This call starts the named task. If the Start Address field is non-zero, it indicates the address at which the task is to be started. If this field contains zero, the task is started at its established transfer address. If this field contains an address which is not within the specified task's memory, no error is detected by SVC 6. However, the specified task aborts as soon as it starts.

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. The UDL of the specified task is used to construct the appropriate Wait call; thus, this function should not be directed at a task which does not use the UDL in the defined manner. Bytes 192-255 of the UDL are used.

This call is rejected if it is self-directed.

Message Rings and Message Buffer Structures

To receive messages sent by other tasks, using SVC 6 Function Code M, the called task must be set up to receive messages. This means that the called task must have in the A(Message Ring) field of its UDL the address of the first of a chain or ring of message buffers. Each buffer must be 76 bytes long. Each buffer must start on a fullword boundary. The first 4 bytes of each buffer contain a link address that points to the next buffer in the chain or ring, or contain zero. The next eight bytes are used for the TASKID of the sending task; this is stored by the system. The remaining 64 bytes are used for data. The most significant bit (bit 0) of the " link address" field must be set up initially to zero.

The called task must also be able to have message parameters added to its Task Queue. This means that bit 19, Enable Queue Entry on Task Message, of its current TSW must be set. See Table 3-1 for TSW contents.

When a message is sent, the appropriate TSW bit is examined. If it is set, the task has a Task Queue, the A(Message Ring) field in the called task's UDL is examined. If this field contains zero or an illegal address, the request is rejected. If this address is valid and non-zero, it is taken to be the address of a fresh buffer.

The link field of the buffer is examined next. If its most significant bit (Bit 0) is set to 1, the system infers that the buffer is full of unprocessed data and the request is rejected. If this bit is set to zero, the system moves the message from the calling task into this buffer, appending the caller's TASKID as a header. Bit zero of the link field is then set to 1, indicating a full buffer. The address of the buffer, with a reason code of 6 is added to the called task's Task Queue. The contents of the link field (with bit zero still set to zero) are placed in the user's A(Message Ring) field in the UDL. Thus, the next Send Message call goes directly to the next buffer, without having to examine any previous buffers in the chain.

The called task, upon receiving the message, takes a trap if the Task Queue Service Trap Enable bit in the TSW is set.

If the system cannot perform the Send Message function, an error code of 11 is given.

Message Buffer Structures

The mechanism for message passing permits the task receiving messages a choice of message buffer structures. Figure 4-3 shows four possible structures that may be used.

Figure 4-3 (a) shows a single-buffer ring. When the buffer is empty, its link field (pointing to itself) is positive. When a message is received by the called task, bit 0 of its link field is set to 1 by the send message routine. All further messages are inhibited until the receiving task sets bit 0 of the link field to zero again.

Figure 4-3 (b) shows a single-buffer chain. When a message is received, the buffer's link field is set to Y'80000000' (i.e., bit 0 is set to a 1) and the contents of A(MESSAGE RING) in the UDL are set to zero (i.e., previous contents of the buffer's link field). In order to receive further messages, the receiving program must set A(MESSAGE RING) to point to an empty buffer in which bit 0 of the link field is set to zero.

A multiple-buffer ring is illustrated in Figure 4-3 (c). Initially, all link fields are set positive. When buffer 1 is filled, A(MESSAGE RING) is set to point to buffer 2. When buffer 2 is filled, A(MESSAGE RING) is set to 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 positive, 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.

A multiple-buffer chain is shown in Figure 4-3 (d). 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.

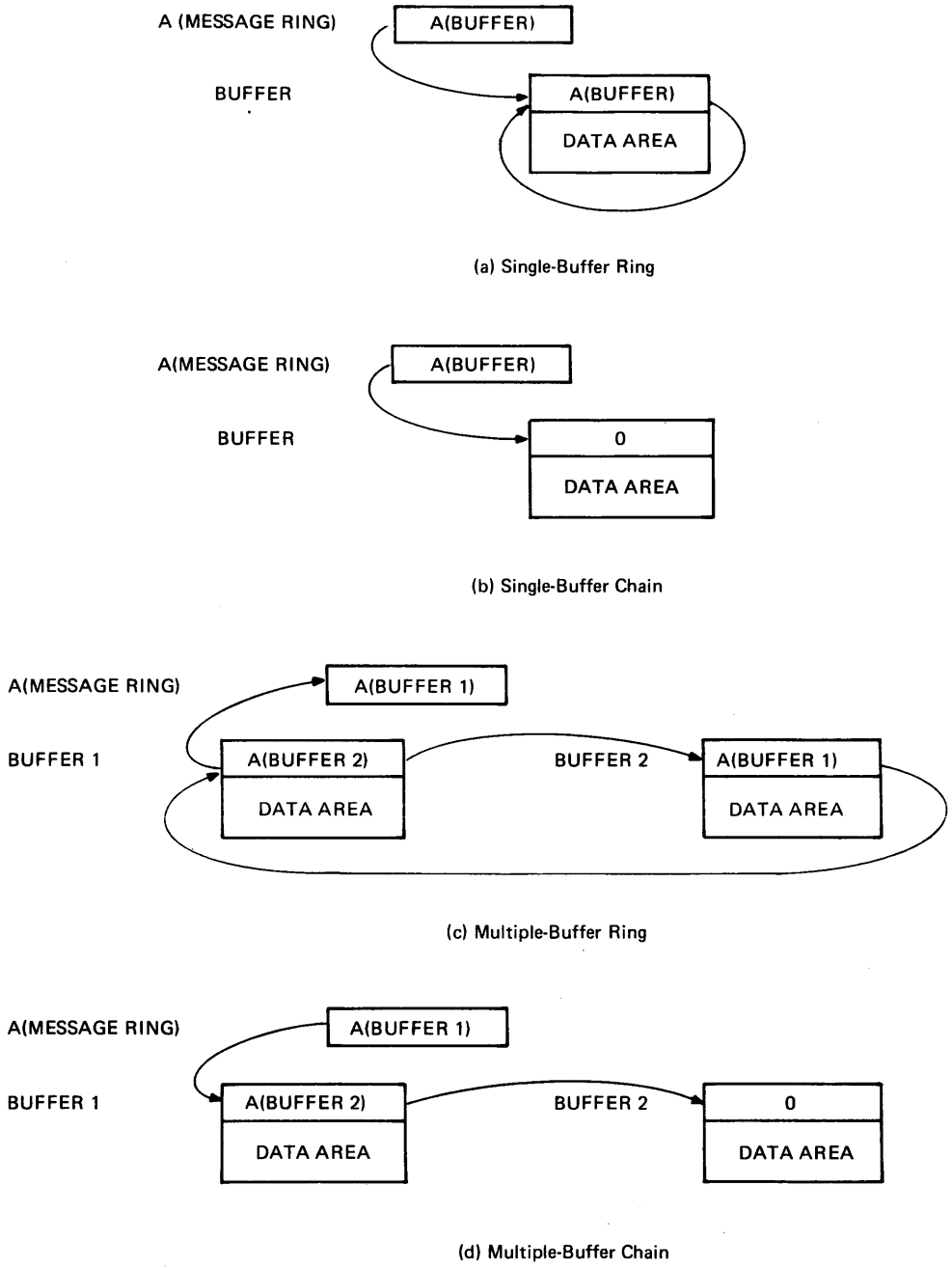


Figure 4-3 Message Buffer Structures

SVC 7 - FILE HANDLING SERVICES

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

- 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 file 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.

SVC 7 also provides the following facilities for use with the INTERDATA Telecommunications Access Method (ITAM):

- Allocation and delection of Line Control Blocks (LCBs) for buffered Terminal Manager Access via SVC 1
- Assignment and closing of Logical Units for Line Drivers (via SVC 15) and Terminal Manager (via SVC 1) access
- Renaming and Reprotecting ITAM lines (SVC 15 access devices), and Terminals (SVC 1 access devices)

The SVC 7 Parameter Block must be on a fullword boundary in a writable segment. Its format is shown in Figure 4-4.

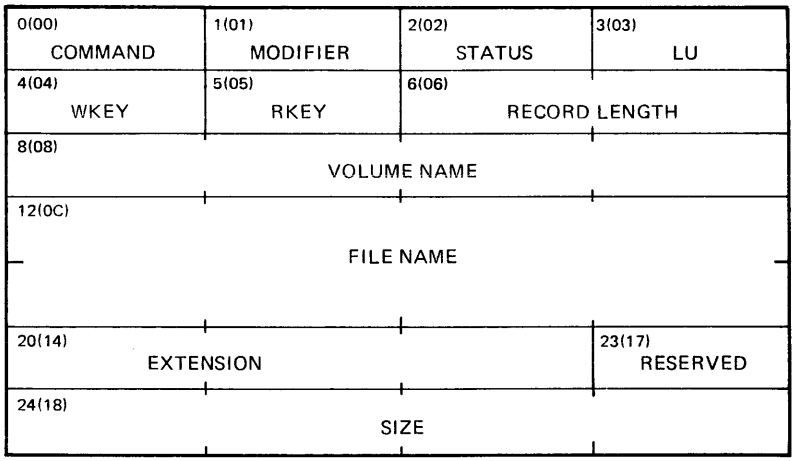


Figure 4-4. SVC 7 Parameter Block

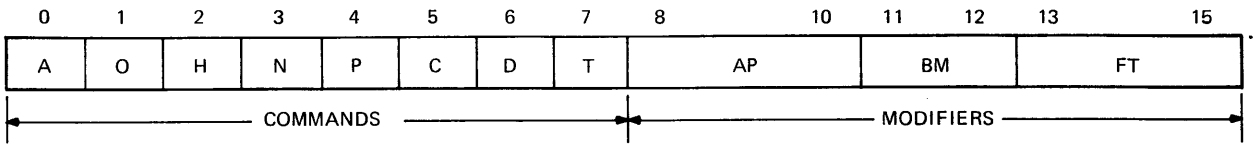
Although not all fields of this parameter block are required for each function of SVC 7, the 28 bytes required for the full parameter block must be reserved. The meaning and use of each field is explained in the description of each function requiring that field. The parameter block may be coded as follows:

```
        ALIGN   4
PARBLK  DB      CMD         COMMAND (1 BYTE)
        DB      MOD         MODIFIER (1 BYTE)
        DS      1           STATUS (1 BYTE)
        DB      LU          LU (1 BYTE)
        DB      WKEY        WRITE KEY (1 BYTE)
        DB      RKEY        READ KEY (1 BYTE)
        DC      H'LRECL'    RECORD LENGTH (2 BYTES)
        DC      C'VOLN'     VOLUME (4 BYTES)
        DC      C'FILENAME' FILE NAME (8 BYTES)
        DC      C'EXT'      EXTENSION (3 BYTES – followed by 1 reserved byte)
        DC      F'SIZE'     SIZE (4 BYTES)
```

SVC 7 – Parameter Block Fields

COMMAND/MODIFIER

The format of the Command/Modifier halfword is shown in Figure 4-5.



A (BIT 0) ALLOCATE; REQUIRES FILE TYPE (FT) FIELD AS MODIFIER.
O (BIT 1) ASSIGN; REQUIRES ACCESS PRIVILEGE (AP) AND BUFFER MANAGEMENT (BM) FIELDS AS MODIFIERS.
H (BIT 2) CHANGE ACCESS PRIVILEGES; REQUIRES AP FIELD AS MODIFIER.
N (BIT 3) RENAME.
P (BIT 4) REPROTECT.
C (BIT 5) CLOSE.
D (BIT 6) DELETE.
T (BIT 7) CHECKPOINT.

ALL BITS ZERO = FETCH ATTRIBUTES

Figure 4-5. SVC 7 Command/Modifier Halfword

If more than one command bit is set, the SVC 7 functions are sequentially processed from left to right.

The modifiers are bits 8-15; they are defined as follows.

Bits 8-10 specify Access Privileges, encoded as follows:

000 = SRO	Sharable Read Only
001 = ERO	Exclusive Read Only
010 = SWO	Sharable Write Only
011 = EWO	Exclusive Write Only
100 = SRW	Sharable Read-Write
101 = SREW	Sharable Read, Exclusive Write
110 = ERSW	Exclusive Read, Sharable Write
111 = ERW	Exclsuive Read-Write

For SVC 7 calls to direct access files, Bits 11-12 specify Buffer Management, encoded as follows:

00 = Default buffer management method
01 = Unbuffered Physical
10 = Buffered, Logical

The default buffer management method is unbuffered physical for Contiguous files and buffered logical for Chained and Indexed files. Current implementations ignore this field and the default is always used.

For SVC 7 calls to ITAM devices, Bits 11-12 specify Access Method, encoded as follows:

00 = Terminal Level (SVC 1) Access
11 = Line Level (SVC 15) Access

Bits 13-15 specify File Type, encoded as follows:

000 - Contiguous
001 = Chained
010 = Indexed
100
.
. - Reserved, considered illegal
.
110
111 = ITAM Buffered Terminal Manager

On a Fetch Attributes call, the Modifier field is not used. Instead, the Device Code is returned in this field.

Error Status

The interpretation of the status byte depends upon the command 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 4-8 for reference.

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

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 direct-access files and devices 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.

Record Length

This halfword field, on an Allocate file call, must contain the logical record length for a buffered file. For an allocate Line Control Block call, this field must contain the logical record length in bytes to be associated with the logical Terminal.

On a Fetch Attributes call, the logical length of a file or physical record length of a device is returned in this field. RECORD LENGTH is not used for other functions.

TABLE 4-8. SVC 7 RETURN CODES

ERROR CODES (HEXADECIMAL)	MEANING
00	NO ERROR
01	ILLEGAL FUNCTION
02	LU ERROR
03	VOLUME ERROR
04	NAME ERROR
05	SIZE ERROR
06	PROTECT ERROR
07	PRIVILEGE ERROR
08	BUFFER ERROR
09	ASSIGNMENT ERROR
0A	TYPE ERROR
0B	FILE DESCRIPTOR ERROR
0C	TGD ASSIGNMENT ERROR
80-FF	I/O ERROR

Volume Name (VOLN) or Device Mnemonic*

This four-byte ASCII field identifies the volume for direct-access devices, or the device mnemonic for non-direct-access devices. This field is required for the Allocate, Assign, Delete, and Fetch Attributes functions.

VOLN together with FILENAME and EXT fields identify 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 ASCII field identifies the file on a direct-access device; it is not required for a non-direct-access device. The user must specify the filename for Allocate, Assign, Rename and Delete calls. The filename is returned by the system on a Fetch Attributes call; it is blank for a non-direct-access device.

This field must also be used to allocate or assign a buffered logical terminal (LCB).

Extension*

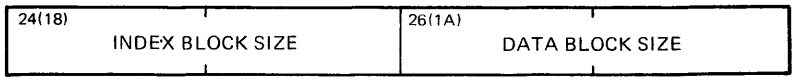
This three-byte ASCII field identifies the file type (e.g., OBJ, TSK, CSS, etc.), on a direct access device. It is treated as an extension of the FILENAME, and is required under the same conditions. The byte following this field is reserved for future expansion and is ignored.

Size

The Size field is defined for the allocate call depending on the type of file being allocated. For a Contiguous file, the Size field must be a fullword containing the file size in sectors. For a Chained file, the Size field must be a fullword containing the physical block size in sectors. For an Indexed file, the Size field is divided into two halfword fields; the first halfword field contains the index block size in sectors, the second halfword contains the data block size in sectors. For ITAM Buffered Terminal Managers, this field contains the physical block size in bytes. On a Fetch Attributes call, this field is used to return the current size of a direct access file; SIZE is not used for a non-direct-access device.

*For full description of the syntax of the VOLN, FILENAME and EXTENSION fields, see section entitled "File Identification".

Note that the full parameter block should be specified for the Fetch Attributes call or the system may overwrite other information. The SVC 7 Parameter Block SIZE field for Indexed files is as follows:



NOTE

If the Index Block size field is specified as zero, the File Manager allocates index blocks of size one sector.

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 Chained and Indexed files, only a directory entry is reserved. The protect keys are entered in the directory. The required parameters are FT, KEYS, LRECL, VOLN, FILENAME, EXT and SIZE. This call is also used to allocate an ITAM Line Control Block for a Buffered Terminal Manager.

Applicable error codes are:

1(01)	Illegal Function; an illegal file type as 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; there is not sufficient space on the specified volume to allocate a file of the specified size
7(07)	Privilege Error; entire disc currently assigned Exclusive Read/Write
10(0A)	Type Error; the specified volume is not a direct-access device
11(0B)	File Descriptor Error
128-255 (80-FF)	I/O Error (as returned by SVC1 defined in Table 4-4)

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 proceeds 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 required 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 WEO (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). Only the keys are checked, for the given access privilege, for non-direct-access devices.

The Assign function is also used to establish a logical connection between an ITAM line or terminal and the task through a Logical Unit. This logical assignment can be used for I/O operations via SVC 1 and SVC 15.

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

Applicable error codes are:

1(01)	Illegal function; invalid BM field
2(02)	LU Error; Illegal LU
3(03)	Volume Error; no such volume
4(04)	Name Error; no such name on given volume
5(05)	Size Error; no room on disc for data or index block
6(06)	Protect Error; mismatch on protection keys
7(07)	Privilege Error; requested privilege may not be granted for file, or entire disc is currently assigned Exclusive Read/Write
8(08)	Buffer Error; no room for FCB or buffer
9(09)	Assignment Error; LU already assigned or off-line
11(OB)	File Descriptor Error
12(OC)	Attempt to assign a Trap generating Device
128-255(80-FF)	I/O Error – Interpreted as SVC 1 error codes defined in Table 4-4

NOTE

A Chained file or ITAM Buffered Terminal Manager requires two single buffers to be allocated in memory, each equal to the physical block size. An Indexed file requires two data buffers, each equal to the data block size, and one index buffer, equal to the index block size.

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 entries in the SVC 7 Parameter Block are required, the LU and AP portions of the modifier field.

The requested new access privilege cannot, however, change the I/O mode which was established when the file was originally assigned. For example, if the file was assigned for ERO, no access privilege requiring write access is allowed. Table 4-9 shows all valid access privilege changes.

TABLE 9. VALID ACCESS PRIVILEGE CHANGES

CHANGE FROM \ CHANGE TO	SRO	ERO	SWO	EWO	SRW	SREW	ERSW	ERW
SRO	Y	Y	N	N	N	N	N	N
ERO	Y	Y	N	N	N	N	N	N
SWO	N	N	Y	Y	N	N	N	N
EWO	N	N	Y	Y	N	N	N	N
SRW	Y	Y	Y	Y	Y	Y	Y	Y
SREW	Y	Y	Y	Y	Y	Y	Y	Y
ERSW	Y	Y	Y	Y	Y	Y	Y	Y
ERW	Y	Y	Y	Y	Y	Y	Y	Y
Y = VALID REQUEST			N = INVALID REQUEST					

If an error is encountered while processing this request, the file remains assigned with its original access privilege. The only legal Access privileges for SVC 15 assignment are those specifying both read and write access.

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
128-255 (80-FF)	I/O Error (as returned by SVC 1)

Rename

This function changes the name of an assigned file. The file must currently be assigned for 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 non-direct access 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 for non-direct access device
11(0B)	File Descriptor Error
128-255 (80-FF)	I/O Error (as returned by SVC 1)

Reprotect

This function changes the protection keys of an assigned file. The required parameters are KEYS and LU. The given LU must be assigned to a direct-access file, which must currently be assigned for ERW (unless the caller is an Executive task which may modify the protection keys of non-direct-access devices). If either the specified Read Key or Write key is equal to X'FF', that key is ignored (unless the caller is an Executive task). If the call is rejected, the previous keys remain unchanged.

Applicable error codes are:

2(02)	LU Error; illegal LU
7(07)	Privilege Error; not assigned for ERW
9(09)	Assignment Error; LU not assigned
10(0A)	Type Error; LU for non-direct-access device
128-255 (80-FF)	I/O Error (as returned by SVC 1)

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 de-assigned. Logical Units assigned for Write access to files or buffered Terminals (SVC 1) have any partially filled buffers written to the file by the CLOSE call.

Applicable error codes are:

2(02)	LU Error
9(09)	Assignment Error; LU Not Assigned
128-255 (80-FF)	I/O Error (as returned by SVC 1)

Delete

For a Delete function, the VOLN, FILENAME, EXT, and KEYS parameters must specify a direct-access file or ITAM Line Control Block 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. The sectors previously occupied by the file on the disc are marked as available in the Allocate Map.

Applicable error codes are:

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; file not closed, or entire disc currently assigned Exclusive Read/Write
10(0A)	Type Error; non-direct-access device
11(0B)	File Descriptor Error
128-255 (80-FF)	I/O Error (as returned by SVC 1)

Checkpoint

The Checkpoint function flushes system Buffer Management buffers and updates the directory entry for a Chained or Indexed file or ITAM Buffered Terminal Manager on a given LU. LU is the only required parameter. Requesting a checkpoint for a Contiguous file or for a non-direct-access device has the same effect as an SVC 1 Wait-only call.

The applicable error codes are:

2(02)	LU Error
9(09)	Assignment Error; LU Not assigned
128-255 (80-FF)	I/O Error (as returned by SVC 1)

NOTE

The user may wish to employ Checkpointing after sensitive data is added to a buffered file. Logical blocking of data in memory in system buffers leaves the file vulnerable. The integrity of the data can be preserved on the direct-access device by Checkpointing. In case of system failure, all data on Chained and 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. File name, access privileges, and keys need not be specified.

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, RECORD, VOLUME NAME, FILENAME, EXT, SIZE and MOD.

The Write Key/Read Key halfword is redefined to receive an Attributes halfword as given in Table 4-10. Any bit set means the device or file supports the corresponding attribute.

TABLE 4-10. 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 RECORD LENGTH 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). The TTY and CRT, which are strictly variable record length devices, normally have RECORD LENGTH set as though they were fixed-record length devices. This is because such a device is normally used in a fixed-record length method. A Contiguous direct-access file is considered to have a variable record length; a Chained or Indexed file is considered to have a fixed record length, which is the logical record length chosen 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 non-direct access device is returned in the File Descriptor portion of the parameter block.

The current size of a direct-access file is returned in the SIZE field; SIZE is unchanged for non-direct-access devices. The number of logical records is returned for a Chained or Indexed file; the number of sectors is returned for a Contiguous file. These sizes are returned as unsigned hexadecimal numbers.

The MODIFIER byte is set to indicate the file or device type. Table 4-11 contains the codes for all supported devices. Refer to the *OS/32 Series General Purpose Driver Manual*, Publication Number 29-384 for additional information.

TABLE 4-11. DEVICE CODES

CODE (DECIMAL)	FILE OR DEVICE TYPE
0	CONTIGUOUS FILE
1	CHAINED FILE
2	INDEXED FILE
16	MODEL 33 TTY KEYBOARD/PRINTER
17	MODEL 35 TTY KEYBOARD/PRINTER
18	NONEDITING CRT ON TTY INTERFACE
21	INTERDATA CAROUSEL 30, 35 (80 COLUMNS)
22	INTERDATA CAROUSEL 30, 35 (132 COLUMNS)
32	MODEL 33 TTY, PALS OR PASLA
34	NONEDITING CRT ON LOCAL PASLA
36	GRAPHIC DISPLAY TERMINAL ON PASLA
37	CAROUSEL 300
38	CAROUSEL 300, ELECTRONIC
48	2.5MB DISC, FIXED PLATTER
49	2.5MB DISC, REMOVABLE PLATTER
50	5MB DISC, FIXED PLATTER
51	5MB DISC, REMOVABLE PLATTER
52	40MB DISC
53	67MB DISC
54	256MB DISC
64	800 BPI MAGNETIC TAPE
65	1600 BPI MAGNETIC TAPE
66	INTERTAPE CASSETTE
80	HIGH SPEED PAPER TAPE READER/PUNCH
81	MODEL 33 TTY READER/PUNCH
82	MODEL 35 TTY READER/PUNCH
83	CAROUSEL 35, PAPER TAPE READER
96	CARD READER WITHOUT HOLLERITH/ASCII TRANSLATION
97	CARD READER WITH HOLLERITH/ASCII TRANSLATION
112	LOW SPEED LINE PRINTER
114	HIGH SPEED LINE PRINTER
128	8 LINE INTERRRUPT MODULE
129	DIGITAL MULTIPLEXOR CONTROLLER
136	REAL TIME ANALOG SYSTEM
137	REAL TIME ANALOG SYSTEM (EXTERNAL CLOCK)

TABLE 4-11. DEVICE CODES (Continued)

CODE (DECIMAL)	FILE OR DEVICE TYPE
138	MINI I/O ANALOG INPUT
139	MINI I/O ANALOG OUTPUT
140	MINI I/O DIGITAL INPUT/OUTPUT MODULE
144	ASYNCHRONOUS COMM. LINE (LINE DRIVER ONLY) (ITAM)
145	MODEL 33 TTY KEYBOARD/PRINTER (ITAM)
146	MODEL 35 TTY KEYBOARD/PRINTER (ITAM)
147	NON-EDITING CRT (ITAM)
148	GRAPHIC DISPLAY TERMINAL (ITAM)
149	CAROUSEL 300 (ITAM)
150	CAROUSEL 300 (ITAM) WITH ELECTRONIC FORMAT CONTROL
160	BINARY SYNCHRONOUS COMM LINE (LINE DRIVER ONLY) (ITAM)
161	IBM 2780 REMOTE JOB ENTRY EMULATION
162	IBM 3780 REMOTE JOB ENTRY EMULATION
163	BISYNC PROCESSOR-TO-PROCESSOR LINK
168	BINARY SYNCHRONOUS COMMUNICATIONS LINE (LINE DRIVER ONLY) ON QSA (ITAM)
169	IBM 3780 RJE EMULATION ON QSA (ITAM)
170	IBM 2780 RJE EMULATION ON QSA (ITAM)
171	BISYNC PROCESSOR-TO-PROCESSOR LINE ON QSA (ITAM)
255	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 user analogue of the machine-level LPSW instruction.

The format of this call is:

```
SVC 9, A(X2)          Load TSW (RX1,RX2)
SVC 9, A(FX2,SX2)     Load TSW (RX3)
```

The effective address of the SVC instruction specifies the location at which the new TSW is to be found 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 only the status of the current TSW is to be changed, a zero LOC field should be used. 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 non-empty) a TSW swap occurs, storing the just loaded TSW in the UDL and loading a new TSW from the UDL.

If the address specified within the SVC 9 call is outside the task's program space, the task is paused with an illegal address in SVC message.

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.

Note that if a task is in Trap Wait with no task traps enabled, it waits indefinitely or until it is cancelled.

Table 4-12 gives a description of the Task Status Word.

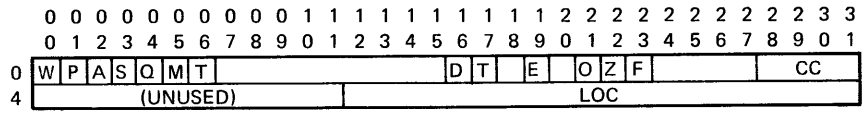
NOTE

A Task Status Word has its value set in one of the following ways:

- An SVC 9 – LOAD TSW is executed
- The TSW option of TET is used when the task is established
- A resident task goes to end of task, in which case the TSW is zeroed.
- A Task Trap occurs and a TSW swap is made.

Merely storing into the UDL does not change the TSW.

TABLE 4-12. TASK STATUS WORD



Task Status Word (2 Fullwords)

BIT	NAME	MASK	MEANING
0	W	Y'80000000'	Trap Wait: task is suspended until a trap occurs.
1	P	Y'40000000'	Power Restoration Trap Enable: Trap is taken on restoration of power following any power failure.
2	A	Y'20000000'	Arithmetic Fault Trap Enable: trap is taken upon arithmetic fault.
3	S	Y'10000000'	SVC 14 Trap Enable: allows SVC 14 service. If this bit is not set, SVC 14 is illegal.
4	Q	Y'08000000'	Task Queue Service Trap Enable: any item added to the Task Queue when this bit is set causes a trap. Also, a trap is taken if a TSW having this bit set is loaded and the Task Queue is not empty.
5	M	Y'04000000'	Memory Access Fault Trap Enable: trap is taken when task attempts to address memory outside partition.
6	I	Y'02000000'	Illegal Instruction Trap Enable: trap is taken when task issues illegal instruction.
16	D	Y'00008000'	Enable Task Queue Entry on Device Interrupt.
17	T	Y'00004000'	Enable Task Queue Entry on Task Call: an SVC 6 Queue Parameter request directed at this task is rejected unless this bit is set.
19	E	Y'00001000'	Enable Queue Entry on Task Message: a message from another task can be received if this bit is set; address of message buffer is added to queue.
21	O	Y'00000400'	Enable Queue Entry on I/O Completion: SVC 1 parameter block address is added to queue upon completion of I/O Proceed.
22	Z	Y'00000200'	Enable Task Queue Entry on Time-out Completion: the parameter in the SVC2 code 23 (time trap) parameter block is added to task queue upon time-out completion.
23	F	Y'00000100'	Enable Queue Entry on SVC 15 Buffer Transfer Command Execution, Termination, or Halt I/O: SVC 15 is only supported by ITAM.
28-31	CC	Y'0000000F'	Current Condition Code, as in PSW.

Bits 7-15, 18, 20 and 24-27 are reserved for future expansion. Bits 0-11 of the second word are unused. Bits 12-31 of the second word contain the current LOC, as in the PSW.

Table 4-13 gives a description of the Task Queue Reason Codes.

TABLE 4-13. TASK QUEUE REASON CODES

CODE (BITS 0-7)	MEANING OF CODE	BITS 8-31
0	Device Interrupt	Parameter associated with device
1	SVC 6 Queue Parameter	Parameter specified in call
6	Message Sent	Address of Message Buffer
8	I/O Proceed Complete	Address of SVC 1 parameter block
9	Timer Termination	Parameter specified in call
10	SVC 15 Buffer	Address of SVC 15 parameter block
11	SVC 15 Command	Address of SVC 15 parameter block
12	SVC 15 Termination	Address of SVC 15 parameter block
13	SVC 15 Halt I/O	Address of SVC 15 parameter block

NOTE

Reason codes not given in Table 4-13 are reserved for implementation in future releases.

SVC 14 – USER SVC

SVC 14 gives the user task a means of accepting an SVC call from a part of itself, e.g., a subroutine or other module.

Its format is:

```
SVC     14, A(X2) or        RX1,RX2 FORMATS
SVC     14, A(FX2,SX2)      RX3 FORMAT
```

The address field of SVC 14 is not interpreted by OS/32, but may be defined by the user task. Normally, it might be used to point to a parameter block.

If the User SVC Trap Enable bit in the task's current TSW is enabled, SVC 14 is enabled. Otherwise, SVC 14 is considered an illegal SVC.

When SVC 14 is executed, if enabled, OS/32 stores the effective program address of the SVC 14 second argument into the SVC 14 Address Pointer location in the task's User Dedicated Locations (UDL). A TSW swap then occurs, using the SVC 14 TSW Swap area in the UDL. The interpretation of this SVC is then left to the user. The effective program address is calculated as for RX1, RX2 or RX3 instructions – see *32-Bit Series Reference Manual,* Publication Number 29-365 or the *Model 8/32 Processor User's Manual,* Publication Number 29-428 for details. This facility permits the user to build a virtual executive within a single task's environment.

It should be noted that OS/32 AIDS (03-064), the OS/32 debugging utility, makes use of SVC 14, and consequently tasks using this facility are not easily debugged through the use of OS/32 AIDS.

SVC 15 – ITAM DEVICE DEPENDENT I/O

This call allows the user task to access ITAM devices at the Device Dependent Level. Refer to the *ITAM/32 Reference Manual* for details.

CHAPTER 5
CONSOLE OPERATIONS

SYSTEM CONSOLE DEVICE

The OS/32 MT system is controlled by the console operator through a device called the system console. This device may be any that is Teletype-compatible such as CRT or Carousel. It has a special relationship to the system; the System Manager task receives command input from the console and writes system messages to it. Tasks may log messages to the system console without reference to its device name.

The system console may be assigned to tasks for ordinary I/O purposes, just as any other device; however, all I/O requests to this device are intercepted by the System Manager, which performs them on behalf of the calling task. If the system console is an ASR TTY or Carousel 35, only its keyboard/printer unit may be used by calling tasks; the reader/punch unit is reserved for the System Manager's use.

Prompts

When the console operator is expected to enter data at the system console, a prompt is output. This prompt takes one of the following forms:

*	(command request)
TASKID>	(data request)

The command request prompt(*) is output whenever the system is ready to accept another command.

The data-request prompt (TASKID>) is output whenever a task is attempting to perform a read I/O request from the system console. The TASKID field of this prompt is the name of the task requesting data. For the background task, the TASKID is ".BG".

The console operator should satisfy the data request as soon as practical, since system messages are held in abeyance until the data request is satisfied.

BREAK Key

If a task is in the process of reading from or writing to the system console, the operator can interrupt this I/O in order to enter a command by depressing the BREAK (or ESC. for some devices) Key of the console device. This forces the system into Command mode for the entry of one command line. After the command line has been accepted, the user I/O to the console is restarted. This process is transparent to the user task.

The BREAK Key may also be used by the operator to terminate further system responses to a command. This is particularly useful in cases such as the EXAMINE and DISPLAY commands, where large quantities of data may be output at the system console.

COMMAND SYNTAX

Commands are accepted one line at a time. A command may not be spread over two or more lines. Multiple commands may appear on the same line, separated by semicolons (;). A command line is terminated by a carriage return.

Commands are composed of:

Mnemonics

Decimal numbers

Hexadecimal numbers

Task identifiers

File descriptors

If multiple commands are specified, they are executed sequentially. If an error is detected, all commands preceding the error are executed and subsequent commands ignored.

Mnemonics

Mnemonics are shown in this manual in upper-case letters. Mnemonics may be abbreviated. This abbreviation may consist of any number of characters from the "minimum abbreviation" to the full mnemonic. Minimum abbreviations are selected to resolve ambiguities between mnemonics while remaining as short as possible. Minimum abbreviations are underlined in this manual, as follows:

REWIND

For example, given the above command:

REW REWI REWIN REWIND	these are all legal forms of the command shown above
RE	illegal, too short
REWA	illegal, misspelled
REWINDZ	illegal, too long

Optional Operands

Some commands have optional operands. These are annotated with brackets surrounding the entire optional part of the command, as follows:

COMMAND aaaa, [bbbb] [,cccc]

In this example, the operand aaaa is not optional, while the operands bbbb and cccc are optional. Note that the comma preceding operand cccc is also optional, but that the comma preceding operand bbbb is required.

ORDER xxxx [,yyyy [,zzzz]]

In this example, operand zzzz must not be entered without operand yyyy. This is shown by the nested brackets. Legal forms of this command are:

OR	xxxx
OR	xxxx,yyyy
OR	xxxx,yyyy,zzzz

Whereas in the previous example, legal forms are:

COM	aaaa,
COM	aaaa,bbbb
COM	aaaa,bbbb,cccc
COM	aaaa,,cccc

General Syntactic Rules

Multiple commands may appear on a line, separated by semi-colons (;).

Certain commands must appear last on a line, or must be the only command on the line. These special commands are discussed in the sections dealing with the individual commands.

If the first character of any command is an asterisk (*), the remainder of that entire line is considered to be a comment and is not executed, although it is copied to the system log device if logging is active.

Decimal and Hexadecimal Numbers

The OS/32 MT command structure uses decimal rather than hexadecimal operands for almost every purpose. The only major exception of this rule is in the case of addresses, which are invariably expressed in hexadecimal.

Most numeric operands are of integer format and do not tolerate the presence of decimal points. The only exception to this rule is in the SET PARTITION command.

Leading zeros may be omitted in numerical operands, whether decimal or hexadecimal.

Task Identifiers

Task identifiers must consist of from one to eight characters; the first character must be alphabetic and those remaining must be alphanumeric.

Thus:

TASK3 FRED X T997XY25	these are all valid task identifiers
34TASK	invalid, first character not alphabetic
T43.2	invalid, non-alphanumeric character
TASK12345	invalid, more than eight characters

The background task has a special identifier, ".BG".

File Descriptors

File descriptors (generally abbreviated fd in this manual) are composed of three fields:

```
voln: filename.ext
```

Voln is the name of the volume on which the file resides or the device mnemonic of a device. It may be from one to four characters, the first character being alphabetic and the remainder alphanumeric.

Filename is the name of the file. It may be from one to eight characters, the first being alphabetic and the remainder alphanumeric.

Ext is the filename extension field. It may be from zero to three characters, which must be alphanumeric.

Voln need not be specified, the default being the system volume. If voln is not entered, the colon (:) separating voln and filename should not be entered.

Ext need not be specified; the default is generally the blank extension, but some commands make use of a different default value. If ext is not entered, the period (.) separating filename and ext should not be entered.

File descriptors may refer to devices as well as to direct-access files. In this case, the voln field is the four-character device mnemonic; filename and ext, if entered, are ignored. The colon following voln must always be entered in this case.

Example of legal file descriptors are:

PACK:FRED.TSK	
FRED.TSK	the same file, if PACK is the system volume
FRED	the same file, if TSK is the default extension
ABC:FOO	default extension, specified volume
CARD:	name of a device

ERROR RESPONSE

If a command input is not acceptable to the system or an error condition is detected while processing a command, an error message is output to the system console device. The general format of the message is:

```
XXXX-ERR  TYPE=YYYY  POS=ZZZZ
```

XXXX is an error descriptor of up to four characters (such as MNEM, ALLO, IO, etc.). Error descriptors are defined in Appendix 3, and are listed with each command description as appropriate.

ZZZZ represents the last command item processed by the System Manager. This field is most useful when multiple commands are entered on one line. It is not always meaningful.

YYYY indicates the type of error encountered and is output only when an I/O, SVC 6 or file handler error is encountered. The possible types of error are:

I/O:	LU	(Illegal or unassigned LU)
	PRTY	(Parity or recoverable error)
	UNRV	(Unrecoverable error)
	EOF	(End of File detected)
	EOM	(End of Medium detected)
	DU	(Device unavailable)
	FUNC	(Invalid function for specified device/file)
FILE:	LU	(Illegal LU)
	VOL	(No such volume/device)
	SIZE	(Erroneous record length/size)
	NAME	(Mismatched FILENAME.EXT)
	PROT	(Mismatched protection keys)
	PRIV	(Mismatched access privilege)
	BUFF	(Insufficient system space to obtain FCB)
	ASGN	(LU not assigned)
	TYPE	(Non-direct-access device or off-line)
	FD	(Illegal File Descriptor syntax)
	FUNC	(Invalid function)
SVC 6:	NLU	(No partition vacant with sufficient LUs)
	PRES	(Specified TASKID already present in system)
	LIB	(Invalid data in Loader Information Block)
	MEM	(No partition of sufficient size vacant)
	IO	(I/O error detected on specified device or file)
	NOFP	(System does not support Floating Point)
	SEG	(RTL or TCOM not present when trying to load task using them)
	NMSG	(task has messages disabled)

If an I/O error occurs during execution of a file management or SVC 6 function, the following error message is output:

```
XXXX-ERR  TYPE=IO  TYPE=YYYY  POS=ZZZZ
```

where YYYY indicates one of the above I/O error types.

The error response to an unrecognized command is:

```
MNEM-ERR
```

All commands following an erroneous command on a command line are ignored.

The message:

```
SEQ-ERR
```

is output if the particular command cannot be accepted because of the state of the system (e.g., the SET PARTITION Command is rejected if there are any active tasks). Such restrictions are discussed in the explanation of the particular commands.

GENERAL SYSTEM COMMANDS

The following commands pertain to the system as a whole. As such, they have a global effect on the system or display global system information:

SET TIME
DISPLAY TIME
VOLUME
SET LOG
DISPLAY MAP
DISPLAY ITAMTERM
SET PARTITION
SET SLICE

Set Time

The SET TIME command should be entered when the system is first loaded and after any power failure. It may be entered at any other time that the system clock is incorrect. The day, month, and year are automatically updated by the system (even during leap years). The format of this command is:

SET TIME mm/dd/yy,hh:nn:ss

where:
mm = month
dd = day
yy = year
hh = hours (24-hour clock)
nn = minutes
ss = seconds

All operands are in decimal. Example:

SE T 2/24/75,3:5:00

Alternatively (by SYSGEN option) the date operand may be entered in the format: dd/mm/yy.

If a SET TIME command is entered while there are uncompleted time intervals (see SVC 2 code 23), the tasks which initiated the incomplete intervals are affected in the following way:

1. Seconds from midnight. The data is updated; this has no effect on any time of day interval even if the date entered differs from the previous date entered. The time difference is used to adjust all seconds from midnight intervals.
2. Milliseconds from now. Elapsed time intervals are unaffected by a change in the time by a SET TIME.

For example, if the current date is 11/22/74 and the current time is 11:50 AM and there are three intervals outstanding:

A. Time of day interval set to complete on 11/22/74 at 2:00 PM.
B. Time of day interval set to complete on 11/23/74 at 8:00 AM.
C. Elapsed time interval set to complete on 11/22/74 at 1:00 PM.

If a SET TIME 11/21/74, 10:50:00 is entered, the time intervals are as follows:

A. Time of day interval set to complete on 11/21/74 at 2:00 PM.
B. Time of day interval set to complete on 11/22/74 at 8:00 AM.
C. Elapsed time interval set to complete on 11/21/74 at 12 NOON.

Possible error responses to SET TIME are:

FORM-ERR Command syntax error; e.g., SET TIME 12/16/74/1:00:00
(Slash instead of comma between date and time)

PARM-ERR Operand syntax error; e.g., SET TIME 68/74/18,1:00:00
(Invalid date specified)

NOPR-ERR Operand missing; e.g., SET TIME 1/1/74
(Time operand missing)

Display Time

The DISPLAY TIME command causes the current date and time to be output to the system console or to a specified file or device. Its format is:

DISPLAY TIME [,fd]

The optional operand fd specifies the file or device to which the display is to be output; if omitted, the display is output to the system console. The display has the following format:

mm/dd/yy hh:nn:ss

or alternatively (by SYSGEN option):

dd/mm/yy hh:nn:ss

Possible error responses to DISPLAY TIME are:

FORM-ERR	Command syntax error; e.g. DISP TIME CCC
PARM-ERR	Operand syntax error; e.g. DISP XXJX
FD-ERR	Invalid file descriptor; e.g. DIS TIME, 135
ASGN-ERR	Output device/file could not be assigned
IO-ERR	I/O error encountered on output device/file

Volume

The VOLUME command is used to set or to change the name of the system volume or to interrogate the system for the current system volume name. Any commands that do not explicitly specify a volume name use the system volume as a default. The format of this command is:

VOLUME [voln]

where voln is a four-character volume identifier. No test is made to ensure that the volume is actually on line at the time the command is entered. If voln is not specified, the name of the current default system volume is output to the console device.

Possible error responses to VOLUME are:

FORM-ERR	Command syntax error; e.g., VOLUME ABCDEF
PARM-ERR	Operand syntax error; e.g., VOLUME 157
NODA-ERR	No direct access support SYSGENed

Set Log

The SET LOG command is used to set the system log device. The system log device receives a copy of all system console I/O. This copy includes:

- All command lines entered from the console or from the Command Substitution System (CSS);
- All responses to these commands (other than prompts);
- All messages logged by tasks.

The format of this command is:

SET LOG [fd [,COPY]]

This copy is produced on a file or device specified by fd.

This device may be changed at any time by another SET LOG command. If no operands are specified, logging is terminated. Logging is automatically terminated under the following conditions:

I/O error on the log device
System initialization
Power restoration

When logging is terminated, the system console device again receives all output.

The SET LOG command may be used for two primary purposes. These are:

– To provide a historical record of system operation, often on a Magnetic Tape or direct access file.

– To allow system output, e.g., displays, log messages, etc., to proceed on a high-speed device rather than on a system console.

If the COPY operand is specified, the system console continues to receive all system console I/O with a copy being sent to the log device; if COPY is not specified, the system console receives only its own error messages.

The Log device may be shared with user task output.

Possible error responses to SET LOG are:

PARM-ERR	Operand syntax error; e.g., SET FOX
FD-ERR	Invalid file-descriptor; e.g., SET LOG PACK.X.Y
ASGN-ERR	Log device/file could not be assigned; e.g., device off-line or assigned for exclusive use to a task
IO-ERR	I/O error occurred on output device/file

Display Map

The DISPLAY MAP command causes a map to be output to the console or to a specified file or device. The display map may be a map of the entire system or of a particular partition or task common segment. The format of the command is:

DISPLAY MAP [/id] [,fd]

The id field can be .BG, .LIB, .TCM, or .SYS to refer to the background partition, the resident library, local task common or system space. It can also be the name associated with an occupied partition or global task common segment. If a TASKID and a task common segment share a common name, the display pertains to the partition and not the task common segment.

The optional operand fd indicates the file or device on which the map is to be output; if omitted the display is output to the system console:

If the optional id is specified, only the map pertaining to the task or segment is output.

A sample format of an entire system map is as follows:

PART	NAME	START	SIZE	STAT	PRI
	.LIB	0C000	12.75		
	.TCM	0F300	11.75		
1	ABC	12200	11.00	A	37
2	XYZ	14E00	8.50	P	126
3		17000	13.00		
4	PDQ35	20400	11.00	RD	13
5		23000	6.50		
6		23A00	13.50		
	.BG	28000	16.00		100
	.SYS	2C000	16.00		
	TSK1	36000	16.00		
	TSK2	46000	80.00		

The NAME field is the name of a task, a partition, or a segment. Where no name is shown, a vacant foreground partition is indicated. The names .LIB, .TCM, .BG, and .SYS indicate the resident library, local task common, background, and system space partitions, respectively. Any name other than these is the name of a foreground task or a global task common segment.

The START field indicates the physical starting address. The SIZE field indicates the size in KB. These sizes are a multiple of .25 KB (i.e., 256 bytes).

The STAT field indicates the status of these tasks, as follows:

D	Dormant
P	Paused (console wait)
A	"Active;" i.e., in any state other than Dormant or Paused

The status may be preceded by an "R" indicating the task is memory resident. A task that is displayed on the map as "Active" may in fact be in a Wait state. The console operator may use the TASK and the DISPLAY PARAMETERS commands to get the actual Wait Status halfword of a given task.

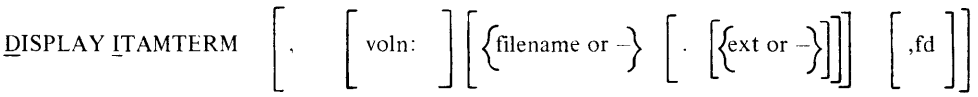
The PRI field indicates the priority in decimal of all tasks currently in foreground or background partitions.

Possible error responses to DISPLAY MAP are:

PARM-ERR	Operand syntax error; e.g., DISP MAD
FD-ERR	Invalid file descriptor specified; e.g., DISP MAP,123:
ASGN-ERR	Output device/file could not be assigned; e.g., device is off-line or assigned for exclusive use to a task.
IO-ERR	I/O error occurred on output device/file

Display ITAMTERM

The DISPLAY ITAMTERM command permits information relative to allocated ITAM Line Control Blocks (LCBs) to be output to the system console or, optionally, to a named file or device. Its format is:



The optional operand fd specifies the device or file on which the display is to be output. If voln is omitted, the default system volume is assumed. If one or more of the optional operands are given, a comma must follow DISPLAY ITAMTERM.

The information displayed is:

Volume Name, Filename and Extension
Device Code (Decimal)
ITAM Extended Device Code (Hexadecimal)
Transmission Block Size (Decimal)
Logical Record Size (Decimal)
Number of Transmission Blocks (Decimal)

Possible error responses to DISPLAY ITAMTERM are:

FORM-ERR	Command syntax error
PARM-ERR	Operand syntax error
NOPR-ERR	Required operand missing
IO-ERR	I/O error encountered on output device.

If a specific ITAM Terminal is not found or if voln has no Line Control Blocks allocated for it, the following message is displayed:

SPECIFIED TERMINAL(S) NON-EXISTENT FOR voln

The following are examples of valid DISPLAY ITAMTERM commands:

1. D I
Displays all ITAM Terminal Line Control Blocks using the default system volume as device mnemonic.
2. D I,,MAGI:
Displays all ITAM Terminal Line Control Blocks using the default system volume as device mnemonic. In this case, the display is routed to device MAGI:.
3. D I, BSCI:
Displays all ITAM Terminal Line Control Blocks using BSCI: as device mnemonic.
4. D I,BSCI:-.-
Is treated identically to 3.
5. D I,BSCI:-.LIN
Displays all ITAM Terminal Line Control Blocks with BSCI: as device mnemonic and extension LIN. If LCB's BSCI:THISTERM.LIN, BSCI:THATTERM.LIN, and BSCI:WHATTERM.LIN exist, all are displayed.
6. D I,SBCL:INPUT.LIN
Displays ITAM Terminal Line Control Block with SBCL as device mnemonic, INPUT as filename and LIN as extension.

Set Partition

The SET PARTITION command is used to vary the size of partitions within the system. It may be entered only when the system is quiesent; i.e., when all partitions are vacant. A dormant, but resident background partition is considered a vacant partition. The format of the command is as follows:

SET PARTITION id/size [,id/size . . . .]

id specifies a partition to be modified. This may consist of:

.LIB
.TCM
.SYS
decimal number

The decimal number is used for vacant foreground partitions; the first foreground partition is partition number 1. The numbers correspond to those displayed by the DISPLAY MAP command.

Size specifies the size to which the associated partition is to be set. These arguments are in decimal KB, in one of the following formats:

n
n.00 n.25
n.5 n.50
n.75

Whenever any partition is adjusted in size, the size of all other partitions, except .BG, remains the same; their starting addresses are adjusted as appropriate. The background partition is expanded or contracted as appropriate.

If .LIB is specified, the size field must be zero; this effectively removes the Library partition. To readjust the size of the Library partition, use the LOAD command (see the section entitled, "Task Related Commands").

For example, assume a system is originally set up as follows:

PART	NAME	START	SIZE	STAT	PRI
	.LIB	0C000	8.00		
	.TCM	0E000	24.00		
1		14000	8.00		
2		16000	8.00		
3		18000	16.00		
4		1C000	16.00		
5		20000	16.00		
	.BG	24000	32.00		
	.SYS	2C000	16.00		

and the console operator enters the following command:

SE PA .TCM/20,3/15.75,4/0,.SYS/18.5

This command changes the size of the local Task Common partition to 20 KB, the third partition to zero KB, the system space to 18.5 KB. The difference, if any, affects the background partition.

A new map is taken and the result is:

PART	NAME	START	SIZE	STAT	PRI
	.LIB	0C000	8.00		
	.TCM	0E000	20.00		
1		13000	8.00		
2		15000	8.00		
3		17000	15.75		
4		1AF00	0.00		
5		1AF00	16.00		
	.BG	1EF00	49.75		
	.SYS	2B600	18.50		

Note the setting of one of the partitions to zero size. This is legitimate; no task can, of course, be loaded into that partition while it has a size of zero.

This command is rejected and partitions remain the same, if it would cause:

- .BG to have a negative size; it is not rejected if it would cause .BG to have zero size.
- The size of .SYS to be decreased to a size less than the amount of .SYS space currently in use by the System Manager.

Possible error responses to SET PARTITION are:

FORM-ERR	Command syntax error; e.g., SET PART 1,8
PARM-ERR	Operand syntax error; e.g., SET PART .XYZ/10
SEQ-ERR	System not quiescent
MEM-ERR	.BG would have negative size or .SYS would be decreased too much

If more than one partition/size is specified in a command, the partitions are adjusted as they appear on the command line. No check is made to see if all requests in the command are valid. If an invalid request is processed, the partitions are left as they were after the previous valid adjustment. In this case, use a DISPLAY MAP command to determine the state of the partitions.

Note that the SET PARTITION command applies only to local memory. The sizes and starting addresses of global task common segments are fixed at SYSGEN time. To modify the global task common segments, the user must re-SYSGEN the system. See the *OS/32 MT Program Configuration Manual,* for details.

Set Slice

The SET SLICE Command is used to invoke the time-slice scheduling option. Its format is:

SET SLICE n

where n is zero or a decimal number greater than or equal to 20 and less than 65,536.

If n is 0, time-slice scheduling is disabled; otherwise, n represents the maximum time, in milliseconds, any one task can remain active if another task of equal priority is ready.

The time slice option is initially disabled.

Refer to the section entitled "Priority and Scheduling" for further information on Time-Slicing.

Possible error responses to SET SLICE are:

NOPR-ERR	Operand n missing from command, e.g., SET SLICE
PARM-ERR	n not 0 or a decimal number within the required bounds, e.g., SET SLICE 5.

UTILITY COMMANDS

This group of commands is useful in debugging or in building Command Substitution System files.

BIAS	BUILD
EXAMINE	ENDB
MODIFY	RESET

Bias

The BIAS command is used to set a base address for the EXAMINE and MODIFY commands. Its format is:

BIAS [{address or *}]

The operand address is a hexadecimal bias, to be added to the address given in any subsequent EXAMINE or MODIFY command. If the operand is omitted, all addresses specified in subsequent EXAMINE and MODIFY commands are treated as unbiased; that is they are assumed to be absolute physical addresses. If an * is specified, the bias is set to be the physical address of the first location of the currently selected task's partition. (See the section entitled "Task Related Commands.")

A BIAS command overrides all previous BIASes.

BIAS is not reset by the RESET command; the operator should enter a BIAS command if the current value is unknown.

Possible error responses to BIAS are:

FORM-ERR	Command syntax error; e.g., BI,*
PARM-ERR	Operand syntax error; e.g., BI FFFFF (too big)
TASK-ERR	* specified, no currently selected task

Examine

The EXAMINE command is used to examine the contents of memory. Its formats are:

(a) EXAMINE address [,n [,fd]]

(b) EXAMINE address [/address [,fd]]

The EXAMINE command, using format (a), causes the contents of the memory location specified by address (as modified by any previous BIAS command) to be displayed. The decimal operand n specifies the number of halfwords to be displayed. If n is omitted, one halfword is displayed.

Using format (b), all data from the first address to the second is displayed in hexadecimal. The BIAS is added to both addresses.

If fd is specified, the EXAMINE command outputs to that specified fd.

All addresses presented are rounded down to halfword boundaries by the system.

The EXAMINE command is used to examine the contents of local or shared memory.

Possible error response to EXAMINE are:

FORM-ERR	Command syntax error; e.g., EXAM 100-200
PARM-ERR	Operand syntax error; e.g., EXA FFFFFF,16; or attempt to examine memory reserved for MAC, or non-existent
FD-ERR	Invalid file descriptor; e.g., EXA 10,10,123
ASGN-ERR	Specified file descriptor could not be assigned, e.g., EKA 10,10,PR:, where PR: exclusively assigned elsewhere.

Modify

The MODIFY command is used to change the contents of memory.

MODIFY address , [data] [,data ...]

causes the contents of the halfword location specified by address (modified by any previous BIAS command) to be replaced with data. The modify address must be aligned on a halfword boundary.

If the operand data is omitted, the modify address has its contents replaced with zeros. Each data field consists of 0-4 hexadecimal digits which represents a halfword to be written to memory starting at the location specified by address. Any string or data less than four characters is right-justified and left-zero filled.

The MODIFY command is used to modify the contents of local or shared memory.

Possible error responses to MODIFY are:

FORM-ERR	Command syntax error; e.g., MOD 124/5
PARM-ERR	Operand syntax error; e.g., MOD 123,0 (not halfword boundary); or address specified not in memory, or reserved for MAC

Build and ENDB

The BUILD and ENDB commands permit the user to copy data from the system console to an arbitrary device or file (these commands may also be entered from a CSS file, see the section entitled "Command Substitution System"). Subsequent lines from the console are not treated as commands, but as data, and are copied to the device or file until an ENDB is encountered. The format of these commands is:

```
BUILD fd
...
ENDB
```

The operand fd is the device or file. *If the fd specified does not contain an extension, then CSS is used as a default.* If a blank extension is desired, the period following the file name must be specified. If fd refers to a non-existent direct-access file, a chained file by that name is allocated, with a logical record length equal to the SYSGENed command buffer length, a blocksize of 1 and keys of 0000.

The BUILD command must be the last command on an input line. Further data appearing on that line is treated as comment and causes no action to be taken.

The ENDB command must appear in the first four characters of the line; any subsequent characters in that line are ignored.

The BUILD command may be entered from the console only if no CSS files are active. The BUILD command may be entered from a CSS file.

Possible error responses to BUILD are:

PARM-ERR	Operand syntax error; e.g., BUILD/
FD-ERR	Invalid file descriptor or no chain file support
ASGN-ERR	Output file/device could not be assigned
SEQ-ERR	CSS file active, build entered from system console
NOPR-ERR	Required fd not specified; e.g., BUILD ;

No error response is possible from ENDB. If ENDB is not entered as the first four characters in the command line, the line is copied to the BUILD file.

Reset

The RESET Command is used to close all background LUs and return the sizes of all partitions to their SYSGENed default. Its format is:

```
RESET
```

This is valid only when the system is quiescent.

Possible error response to RESET is:

SEQ-ERR	System not quiescent

TASK RELATED COMMANDS

The following commands are related to particular tasks:

TASK	ASSIGN
START	DISPLAY LU
PAUSE	CLOSE
CONTINUE	OPTIONS
CANCEL	SET PRIORITY
LOAD	DISPLAY PARAMETERS
	SEND

Task

The TASK command is used to set the currently-selected task, either foreground or background. Task-related commands operate on the currently-selected task and on no other. The format of the command is:

```
TASK [taskid]
```

where taskid is the name of some foreground task in the system, or is the name of the background task, ".BG". If taskid is not specified, the TASKID of the currently selected task is output to the console device.

All task related commands except LOAD are affected by the TASK command. Also affected are the Magnetic Tape and File Control Commands, Format (b) as described in the section entitled, "Device and File Control Commands."

Some CSS commands are affected by TASK; they are described in the section entitled, "Command Substitution System."

For example:

T ABC	Set current task = ABC
CL 2,3,4	Close ABC's LUs 2,3,4
AS 2,CARD:	Assign ABC's LU 2 to device CARD
T XYZ	Set current task = XYZ
OPT RES	Make Task XYZ Resident
CAN	Cancel task XYZ
ST 100	Start task XYZ at relative address 100
T .BG	Set current task = background
PAUSE	Pause the background task

NOTE

When CSS is started, the current value of TASK is associated with the CSS file as the currently-selected CSS task. If a CSS file executes a TASK command it affects only that CSS file's commands, and does not change the value of TASK associated with the console.

If the currently selected task is deleted from the system or if no TASK command has been entered, task related commands (other than LOAD or TASK) are rejected with a TASK-ERR.

Possible error responses to TASK are:

PARM-ERR	Operand syntax error; e.g., specified TASKID not present
TASK-ERR	No currently selected task

Start

The START command is used to initiate task execution. The currently-selected task is started, if it is dormant or paused, otherwise the command is rejected. The format of this command is:

START [address] [, args to prog]

The START command must be the last command on its input line. The operand address represents the address at which the program is to be started. For User tasks this is not a physical address, but is an address within the task's own program. For Executive tasks, it is a physical address within the task partition. If address is omitted, the currently-selected task is started at the transfer address specified when the task was established.

The optional field 'args to prog' contains arguments that are to be passed to the task for its own decoding and processing. All characters between the comma and the line terminator (semi-colon or carriage return) are moved to memory beginning at UTOP. The characters are terminated in memory by a carriage return. If this operand is omitted, a carriage return is stored at UTOP. If there is not enough memory between UTOP and CTOP to pass all the characters, the call is rejected with an ARGS-ERR.

Possible error responses to START are:

FORM-ERR	Command syntax error; e.g., START 100/5
SEQ-ERR	Task active
TASK-ERR	No currently selected task
ARGS-ERR	Insufficient memory between UTOP and CTOP to pass all args to prog.

Examples of valid START commands are:

ST	138	START TASK AT X 138
ST	100,NOSEQ,SCRAT	START TASK AT X 100 AND PASS NOSEQ,SCRAT TO THE PROGRAM
ST	,1000,ABC	START TASK AT TRANSFER ADDRESS AND PASS 1000,ABC TO THE PROGRAM

Pause

The PAUSE command causes the currently selected task to pause as though it has issued an SVC 2 CODE 1, PAUSE (See the section entitled SVC 2 – General Service Functions). Its format is:

PAUSE

Any I/O Proceed ongoing at the time the task is paused is allowed to proceed to completion. If the task is in any Wait state at the time the PAUSE command is entered, all external wait conditions must have been satisified before the PAUSE becomes effective. This command is rejected if the task is dormant or paused at the time it is entered.

Possible error responses to PAUSE are:

FORM-ERR	Command syntax error
SEQ-ERR	Task paused or dormant
TASK-ERR	No currently selected task

Continue

CONTINUE causes a task which has executed a PAUSE SVC 2 CODE 1 or has been paused by the operator to resume operation. The format of this command is:

CONTINUE

Possible error responses to CONTINUE are:

FORM-ERR	Command syntax error
SEQ-ERR	Task not paused
TASK-ERR	No currently selected task

Cancel

The CANCEL command terminates a task as if it had executed an SVC 3,255. The format of this command is:

CANCEL

If the task is non-resident, it is removed from the system; all outstanding I/O is terminated and the task's LUs are closed. If the task is resident, it is not removed from the system; its LUs are not closed, but are checkpointed. This command may be entered even when the currently-selected task is Dormant. It has no effect on a resident task that has already gone to EOT, unless preceded by an OPTIONS NONRESIDENT command. It may be used to remove a non-resident task, which has been loaded but not started, from the system. The normal response to this command is:

hh:mm:ss	TASKID:END OF TASK 255

Possible error responses to CANCEL are:

FORM-ERR	Command syntax error
SEQ-ERR	Task not loaded
TASK-ERR	No currently selected task

Load

The LOAD command is used to load a task. The task is loaded into a specified partition or the first foreground partition large enough to accept the task. The loader searches for a vacant position starting from the partition with the lowest memory address. The command may also be used to load a background task, or a new copy of the Resident Library. The format of this command is:

LOAD taskid [, [fd] [,n]]

The taskid field specifies the name to be assigned to the task to be loaded. The optional fd field specifies the file or device from which the task is to be loaded. If fd is omitted, the file is defaulted to taskid.TSK. If fd specifies a direct access file and no extension is specified, the extension is defaulted to .TSK. An fd with blank extension must have the period after the specified file name. This command is not affected by the TASK command, but may be entered at any time. The optional operand, n, specifies a particular vacant partition in which to load the task.

The taskid field may be one of the following:

- A valid task identifier which is not the same as any task identifier presently in the system;
- .BG
- .LIB

If the first option is selected, the task is loaded into a foreground partition. It then becomes a member of the foreground system and can be selected by the TASK command.

If the second option (.BG) is selected, the task is loaded into the background partition. This command is only accepted when the background partition is vacant, or when the task in the background partition is dormant.

The operand, n, is invalid, if .BG is specified.

The taskid .LIB is used to load the resident library. To do this, all partitions, both foreground and background, must be vacant. When this command is entered, the current resident library, if any, is deleted. The new resident library is loaded from the device specified by fd. This may cause a change in the size of the resident library partition. The bounds of all other partitions in the system are adjusted accordingly. This causes an adjustment in the size of the background partition (see the section entitled "General System Commands"). Any data previously stored in the local task common partition must at this time be presumed lost. The operand, n, is invalid.

Some examples of the LOAD command are:

LO ABC,PTRP:,2	LOAD a task from a device named PTRP: into partition 2. Associate the name ABC with the task.
LO .BG,VOL:CAL.TSK	LOAD the task from file VOL:CAL.TSK into the background partition.
LO .LIB,VOL:FORT.RTL	LOAD the library from file VOL:FORT.RTL.
L T1	LOAD a task from file named T1.TSK on the default system volume into a foreground partition. Associate the name T1 with the task.
L T1.T1.	LOAD a task from the default system volume from a file named T1. into a foreground partition. Associate the name T1 with the task.
LO T3,,4	LOAD the task T3 from file T3.TSK into partition 4.

Possible error responses to LOAD are:

FORM-ERR	Command syntax error; e.g., LOAD,PTRP:
PARM-ERR	Operand syntax error; e.g., LOAD T1, PTRP:,ABCD
PART-ERR	Specified partition not vacant
TKID-ERR	Invalid TASKID syntax; e.g., LOAD 1,PTRP:
ASGN-ERR	Specified fd not assigned for reason denoted by TYPE field
FD-ERR	Invalid file descriptor; e.g., LOAD ABC,A:B:C
LOAD-ERR	Load failed for reason denoted by TYPE field

LOAD error TYPE field:

SEG	(Proper .LIB not in system; task uses TCOM, not in system.)
NLU	(No partition with sufficient LUs vacant.)
PRES	(Specified TASKID already present in system.)
LIB	(Invalid data in Task Loader Information Block.)
MEM	(No partition big enough.)
IO	(I/O error detected on specified device or file.)
NOFP	(System does not support the Floating Point options required by the task.)

Assign

The ASSIGN command assigns a device, file, or ITAM device to one of a task's Logical Units. The format of this command is:

ASSIGN lu, fd [,[access-priv][,keys]]

where lu is the LU number in decimal, fd is the File Descriptor signifying the device or file to be assigned, access-priv is the desired access privilege, and keys signifies the write-read protection keys of the file or device.

Access-priv may contain one of the following:

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

If access-priv is omitted, SRW is assumed. The command is rejected if the requested access privilege cannot be granted.

Keys is a four digit hexadecimal number. The left two digits signify the write-protection key and the right two digits the read-protection key. If omitted, the default is 0000. These keys are checked against the appropriate existing keys for the file or device; the command is rejected if the keys are invalid.

An assigned direct-access file is positioned at the end of the file for access privileges SWO and EWO; it is positioned at the beginning of the file for all other access privileges.

This command is rejected if the specified LU is assigned and the currently selected task is not dormant. To reassign an LU for an active task, the LU must first be closed.

Possible error responses to ASSIGN are:

FORM-ERR	Command syntax error; e.g., AS 1/CR:
PARM-ERR	Operand syntax error; e.g., AS CR:,1
NODA-ERR	No direct access support in this SYSGEN
FD-ERR	Invalid file descriptor; e.g., AS1, PACK:ABC:TSK
LU-ERR	Invalid LU number or LU assigned
PRIV-ERR	Invalid access privilege mnemonic; e.g., ASSIGN 1, CR:,SWO
ASGN-ERR	The assign failed for reason denoted by the TYPE field
TASK-ERR	No currently selected task
SPAC-ERR	Task would exceed established maximum system space usage

Display LU

The DISPLAY LU command permits the operator to display all assigned Logical Units of the currently-selected task. Its format is:

DISPLAY LU [,fd]

where the optional operand fd signifies the file or device on which the display is to output. If the optional operand is omitted, the display is output to the system console.

An example of the DISPLAY LU output is:

```
LU    FD/NAME
01    CR:
02    VOLN:ABC.OBJ
03    PR:
04    VOLN:SCRATCH.
05    TTY:
10    NULL:
```

Possible error responses to DISPLAY LU are:

FORM-ERR	Command syntax error; e.g., DISP LU CCC
PARM-ERR	Operand syntax error; e.g., DISP LUPR:
FD-ERR	Invalid file descriptor; e.g., DISP LU,123
IO-ERR	I/O error detected on output device/file
TASK-ERR	No currently selected task

Close

The CLOSE command permits the operator to close (deassign) one or more files or devices assigned to the currently-selected task's Logical Units. The formats of this command are:

CLOSE [lu [,lu . . . .]]

CLOSE ALL

where the lu operands are decimal numbers signifying the Logical Units to be deassigned. If ALL is specified, all the LUs of the currently-selected task are closed.

Closing an unassigned LU does not produce an error message. A CLOSE Command may only be entered if the referenced task is dormant or paused.

Examples of the CLOSE command are:

CL 1,3,5	Close LUs 1,3, and 5 of the currently selected task
CLOSE A	Close all LUs of the currently selected task

Possible error responses to CLOSE are:

FORM-ERR	Command syntax error; e.g., CLO 1/2/3
CLOS-ERR	Close failed for reason denoted by TYPE field
TASK-ERR	No currently selected task
SEQ-ERR	Task not dormant or paused

Options

The OPTIONS command is used to specify or to change certain options of the currently selected task. An OPTIONS command may be entered if the referenced task is dormant or has been paused. The format of this command is:

OPTIONS opt [,opt . . .]

where opt may be any of the following options:

AFCONT	If the AF trap enable bit is set, a trap is taken; otherwise, the task continues after arithmetic fault with message logged
AFPAUSE	PAUSE after any arithmetic fault
RESIDENT	Task is memory-resident
NONRESIDENT	Task is to be removed from memory at EOT
FLOAT	Task requires single precision floating point registers
DFLOAT	Task requires double precision floating point registers
NOFLOAT	Task requires no floating point registers.
SVCPAUSE	Treat SVC 6 as illegal SVC (applies to .BG only)
SVCCONT	Treat SVC 6 as NOP (applied to .BG only)

NOTE

Unless otherwise specified with the TET/32 OPTIONS command, the default options when a task is loaded are: AFPAUSE, NONRESIDENT, NOFLOAT, SVCPAUSE.

Note that most options are paired, but options are entered in any order and if both members of a pair are entered, the latest one entered is accepted. Thus:

OPTIONS R,NONR

specifies NONRESIDENT.

The AFPAUSE and AFCONT options are normally set up at task establishment time, but may be modified by the console operator. Note that the sequence OPT NON; CANCEL always causes the currently-selected task to be removed from memory. The sequence OPT R; CANCEL always causes the currently-selected task to enter the Dormant state.

The SVCPAUSE and SVCCONT options apply only to the background task; they are ignored if specified for a foreground task.

Possible error responses to OPTIONS are:

PARM-ERR	Operand syntax error; e.g., OPT NO
TASK-ERR	No currently selected task
NOFP-ERR	Floating Point option not supported in OS
SEQ-ERR	Task not dormant or paused

If an operand is invalid, previous valid operands in the same command are processed. In this case use the DISPLAY PARAMETERS command to verify the state of the task options.

Set Priority

The SET PRIORITY command is used to modify the priority of the currently-selected task. Its format is:

SET PRIORITY n

where n is a decimal number from 10 to 249 inclusive. The priority of the currently-selected task is set to n, subject to the following restriction:

> If the task is a foreground task, its priority may not exceed the maximum priority set up at task establishment time. In order to increase this priority, it is necessary to reestablish the task. The maximum priority a background task may attain is set at SYSGEN time. To increase this priority, the system must be re-SYSGENed.

Possible error responses to SET PRIORITY are:

FORM-ERR	Command syntax error; e.g., SET PRI,12
PARM-ERR	Operand syntax error; e.g., SET PRI 0
NOPR-ERR	Required operand n omitted; e.g., SET PRI;
TASK-ERR	No currently-selected task

Display Parameters

The DISPLAY PARAMETERS command is used to display certain parameters pertinent to the currently-selected task. The display appears on the console device, or alternatively on a device or file selected by the operator. The format of this command is:

DISPLAY PARAMETERS [,fd]

Parameters displayed are:

TASK	taskid	task name
CTSW	xxxxxxxx	status portion of current TSW
CLOC	xxxxx	loc portion of current TSW
STAT	xxxxx	task's Wait status
TOPT	xxxxx	task options
CTOP	xxxxx	task CTOP
UTOP	xxxxx	task UTOP
UBOT	xxxxx	task UBOT
SLOC	xxx	task starting location
NLU	xx	number of LUs (decimal)
MPRI	xxx	maximum priority (decimal)
SVOL	xxxx	default volume ID.

The addresses displayed as CTOP, UTOP, UBOT, and SLOC are not physical addresses, but are addresses within the task's own program space. The CLOC may be a program space address or a physical address in a system subroutine being executed on behalf of the task. NLU is given in decimal. SVOL is the ASCII System Volume ID. As such it is not specifically related to the currently-selected task, but it is given here for operator convenience.

TOPT is given in hexadecimal; the definition of task option bits is:

Bit	Mask	Meaning
0	8000	0 – User task; 1 – E-task
1	4000	0 – AFPAUSE; 1 – AFCONT
2	2000	0 – NOFLOAT; 1 – FLOAT
3	1000	0 – NONRESIDENT; 1 – RESIDENT
4	0800	0 – NOTCOM; 1 – TCOM
5	0400	0 – NOLIB; 1 – LIB
6	0200	0 – SVCPAUSE; 1 – SVCCONT
7	0100	0 – NOFLOAT; 1 – DFLOAT

STAT is given in hexadecimal; the definition of wait status bits is:

Bit	Mask	Meaning if set
0	8000	I/O Wait
1	4000	Connection Wait
2	2000	Console Wait (Paused)
3	1000	Load Wait
4	0800	Dormant
5	0400	Trap Wait
6		Reserved
7	0100	Task Wait
8	0080	Time Wait
9–15		Reserved

TSW is given in hexadecimal. For a definition of the status portion of the TSW, see the section entitled Interrupts and Traps.

Possible error responses to DISPLAY PARAMETERS are:

FORM-ERR	Command syntax error; e.g., DISP PARM CCC
PARM-ERR	Operand syntax error; e.g., DISP XYZ
IO-ERR	I/O error detected on output device or file
TASK-ERR	No currently selected task
ASGN-ERR	Invalid File Descriptor; e.g., DISP PA,ABCDE:

Send

The SEND command is used to send a message to the currently-selected task. The format of the command is as follows:

SEND (Up to 64 ASCII characters)

The message is passed to the selected task in the same manner as an SVC 6 Send Message. Following standard SVC 6 procedures, the message consists of an 8 byte TASKID identifying the System Manager, followed by the operator supplied character string.

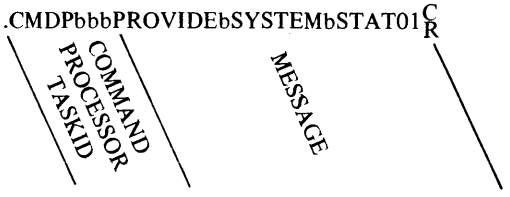
The message data passed to the selected task begins with the first non-blank character following SEND and ends with a carriage return or semicolon as a line termination.

The receiving task must have intertask message traps enabled in its TSW and must have established a message buffer area.

For example:

```
TASK STATS
SEND PROVIDE SYSTEM STAT01 C/R
```

provides the following message for the task STATS:



```
TA TPSRCH
SEN  STOP TAPE SEARCH<CR>
```

provides the following message for the task TPSRCH:

The possible error responses to SEND are:

TASK-ERR	No currently selected task.
NOPR-ERR	No message was provided. The first non-space character following the SEND command was a carriage return.
ARGS-ERR	A message exceeding 64 characters was provided.
SEQ-ERR	Task paused, not yet started or otherwise not capable of receiving a message
SVC6-ERR	An SVC-6 error 11 was returned indicating that the task could not receive a message trap.

Refer to the section entitled SVC 6 Intertask Coordination for more information about SEND MESSAGE.

DEVICE AND FILE CONTROL COMMANDS

The following set of commands is used for device and file control. These commands are not affected by the setting of the currently-selected task:

ALLOCATE	FRECORD
DELETE	FFILE
RENAME	BRECORD
REPROTECT	BFILE
DISPLAY FILES	WFILE
MARK	REWIND
DISPLAY DEVICES	RW

Allocate

The ALLOCATE command is used to create a direct-access file or allocate an ITAM Line Control Block for a Buffered Terminal Manager. The following formats exist for this command:

(a) ALLOCATE fd, CHAINED [, [lrecl [/bsize]] [,keys]]

(b) ALLOCATE fd, CONTIGUOUS, fsize [,keys]

(c) ALLOCATE fd, INDEX [, [lrecl] [/ [bsize] [/isize]] [,keys]]

(d) ALLOCATE fd, ITAM [,lrecl [/bsize] [,keys]]

The operand fd identifies the file to be allocated. Format (a) is used to allocate a Chained file; format (b) is used to allocate a Contiguous file; format (c) is used to allocate an Indexed file; format (d) is used to allocate an ITAM device.

If CHAINED is chosen, the next operand, lrecl, is optional and specifies the logical record length. It cannot exceed 65,535 bytes. Its default is 126 bytes. It may optionally be followed by a slash mark (/) which delimits lrecl from bsize. The bsize operand specifies the physical block size, in 256-byte sectors, to be used for buffering and de-buffering operations on the file. If bsize is omitted, the default value is 1 sector (256 bytes). Note that, in order to assign this file, sufficient room must exist in system space for two buffers, each of the stated size. Therefore, if bsize is very great, the file may not be assignable in some memory-bound situations. At SYSGEN time, a maximum block size parameter is established in the system; bsize cannot exceed this constant. In no case may bsize exceed 255. Both lrecl and bsize are specified as decimal numbers.

If CONTIGUOUS is chosen, the file size operand, fsize, is required and specifies the total allocation size in 256 byte sectors. This size may be any value up to the number of contiguous sectors existing on the specified volume at the time the command is entered. Fsize is specified as a decimal number.

If INDEX is chosen, the operands lrecl and bsize are defined as for CHAINED. The isize optional operand specifies the index block size in decimal. If isize is omitted, the default value is 1 sector (256 bytes). Like bsize, isize cannot exceed the maximum block size which is established at SYSGEN time, and in any case neither can exceed 255.

If ITAM is chosen, the optional operand lrecl specifies the logical record length. The bsize operand specifies the physical block size in bytes.

The last operand. keys, is optional. This operand specifies the write and read protection keys for the file. These keys are in the form of a hexadecimal halfword, the left byte of which signifies the write key and the right byte the read key. If this parameter is omitted, both keys default to zero.

Examples of the ALLOCATE command:

```
AL  THISFILE,CH
```

allocates on the system volume a Chained file named THISFILE, (blank extension) with a logical record of 126 bytes, a buffer size of 1 sector, and protection keys of zero.

```
AL  PROGRAM.TSK,CO,64
```

allocates on the system volume a Contiguous file named PROGRAM.TSK, whose total length is 64 sectors (16KB) and protection keys are zero.

```
AL  FRED:EXAMPLE.OBJ,CH,126
```

allocates on the volume FRED a Chained file named EXAMPLE.OBJ, whose logical record length is 126 bytes. The buffer size of this file defaults to one sector; the protection keys default to zero.

```
AL  MORT:GREATBIG.BLK,CH,132/4
```

allocates on the volume MORT a Chained file named GREATBIG.BLK, whose logical record length is 132 bytes, using a physical block size of 4 sectors. The protection keys default to zero. Note that whenever this file is assigned, the system must have 2KB of available system space (twice the physical block size) for buffers.

```
AL  SAM:DATABASE.X,CH,480,AA44
```

allocates on the volume SAM a Chained file named DATABASE.X, whose logical record length is 480 bytes, physical block size is 1 sector, write protection key is X'AA' and read key is X'44'. Note that the logical record length is permitted to exceed the physical block size.

```
AL  THISFILE,IN,256/4/2
```

allocates on the default system volume an Indexed file named THISFILE (blank extension) with a logical record of 256 bytes, a data block size of 4 sectors, an index block size of 2 sectors, and protection keys of zero.

```
AL  PACK:TEST.OBJ,IN,126//3
```

allocates on the volume PACK an Indexed file named TEST.OBJ with logical record length of 126 bytes. The data block size is defaulted to 1 sector, the index block size is 3 sectors and the protection keys default to zero.

Possible error responses to ALLOCATE are:

FORM-ERR	Command syntax error
PARM-ERR	Operand syntax error
NOPR-ERR	Required operand missing
ALLO-ERR	Allocation failed; for reason denoted by TYPE field
FD-ERR	Invalid file descriptor
NODA-ERR	Direct-access support not included in this SYSGEN

Delete

The DELETE command is used to delete a direct access file. Its format is:

DELETE fd [,fd . . .]

where fd identifies the file to be deleted. To be deleted, the file must not be currently assigned to any LU of any task. This command is not recognized if there are no direct-access devices in the system.

Possible error responses to DELETE are:

FORM-ERR	Command syntax error; e.g., DEL, PACK:A
FD-ERR	Invalid file descriptor; e.g., DEL A:1
NOPR-ERR	Required operand missing; e.g., DEL;
DELE-ERR	Delete failed for reason denoted by TYPE field
ASGN-ERR	fd currently assigned
NODA-ERR	Direct Access support not included in this SYSGEN

Rename

The RENAME command is used to change the name of an unassigned direct-access file or a device. Its format is:

RENAME oldfd, newfd

Examples:

```
REN VOL1:MYFILE.CUR,MYFILE.OLD
REN MT01:,MT02:
```

The volume ID field of the new File Descriptor (newfd) may be omitted for direct-access files. If it is entered, the system ignores it. This command cannot be used to rename a direct-access volume; the OS/32 Disc Initializer must be used for this. Attempts to rename the console device or the null device are rejected with ASGN-ERR and NULL-ERR respectively. An attempt to rename a device to an existent device or volume name is rejected with NAME-ERR.

Possible error responses to RENAME are:

FORM-ERR	Command syntax error; e.g., REN A:B C:D
NOPR-ERR	Required parameter missing, e.g., REN CR:
ASGN-ERR	fd currently assigned
RENM-ERR	Rename failed for reason denoted by TYPE field
NULL-ERR	Attempt to rename the Null device
NAME-ERR	Duplicate Device or Volume Name exists

Reprotect

The REPROTECT command permits the operator to modify the protection keys of an unassigned direct-access file or device. Its format is:

REPROTECT fd, keys

where fd is the name of the file or device and keys is a hexadecimal halfword whose left byte signifies the new write keys and whose right byte signifies the new read key.

Possible error responses to REPROTECT are:

FORM-ERR	Command syntax error; e.g., REPR CR:/FF12
PARM-ERR	Operand syntax error; e.g., REPR CR:,XXYY
ASGN-ERR	fd currently assigned
NOPR-ERR	Keys not specified; e.g., REPR CR:
REPR-ERR	Reprotect failed for reason denoted by TYPE field

Display files

The DISPLAY FILES command permits information from the directory of one or more direct-access files to be output to the system console or optionally to a named file or device. Its format is:

DISPLAY FILES [, [voln:][{filename or -} [. [{ext or -}]]] [,fd]]

The optional operand fd specifies the device or file for the display. If voln is omitted, the default system volume is assumed. If any of the optional operands is given they must be preceded by at least one comma.

The following forms are all acceptable:

1. D F
   displays all files on the default system volume to the console device.
2. D F , , MAG1:
   display all files on the default system volume. In this case, the display is routed to the device MAG1:
3. D F,PACK:
   displays all files on PACK to the console device.
4. D F , PACK: – . –
   is treated identically to 3.
5. D F,SCRT: – . TSK
   displays information concerning all files on SCRT with extension TSK, regardless of filename. If SCRT contains files names THISFILE.TSK, THATFILE.TSK, and LOADER.TSK, all are displayed.
6. D F , – . , PRIN:
   displays information concerning all files on the default system with blank extensions regardless of filename. In this case, the display is routed to device PRIN:
7. D F , – . –
   is treated the same as 1.

The information displayed is:

File Name and Extension
TYPE
LENGTH
KEYS
START/NLR

For Contiguous files, TYPE is CO, the LENGTH is the number of sectors allocated to the file in decimal and START/NLR is the starting sector number in hexadecimal.

For Chained or Indexed files, TYPE is CH or IN respectively, the length is the logical record length in decimal, and START/NLR is the number of logical records in hexadecimal.

Possible error responses to DISPLAY FILES are:

FORM-ERR	Command syntax error
PARM-ERR	Operand syntax error
NOPR-ERR	Required operand missing
IO-ERR	I/O error encountered on output device or file
NODA-ERR	Direct-access support not included in this SYSTEM.

If VOLN has no files on it, the following message is displayed:

NO DIRECTORY ENTRIES ON VOLN

If a specific file is not found, the following message is displayed:

FILENAME NOT FOUND

Mark

The MARK command is used to take a device off-line, or to bring on-line a device that was previously off-line. The format of this command is:

MARK fd, { ON [,PROTECT] / OFF }

The mnemonic name of the device is specified by fd. If the device is the system console device, is the null device, is assigned, was not properly marked off-line, or has any assigned files (if a direct-access device), the command is rejected. After marking on a direct-access device, the volume name associated with it is output to the console device, in the format VOLN=xxxxx.

While a device is off-line, it cannot be assigned to any U-task. E-Tasks are permitted to assign off-line devices.

If the device being MARKed ON or OFF is a direct-access device, the fd used in the command is not the volume identifier, but the actual device mnemonic. For example, to MARK OFF a disc name DSC1 which currently contains a volume named SYS1, the operator enters:

```
MA   DSC1:,OFF
```

This action removes the volume SYS1 from the system. The volume may now be changed, if DSC1 is a removable-cartridge disc. To make the new volume known to the system, the operator enters:

```
MA   DSC1:,ON
```

This causes the Volume Descriptor (see the Section entitled "Files and Devices") of the pack on DSC1 to be read. The new volume ID is made known to the system. The volume ID associated with DSC1 is output to the system console in the format voln=xxxx. *Removable cartridges should not be mounted or dismounted from the system without using the MARK command.*

If a volume is dismounted without being MARKed OFF-line, it can only be MARKed ON-line in the write protected state. The volume cannot again be MARKed ON with write privileges until the Disc Integrity Utility, Revision 05 or later, is run. This ensures that the disc is returned to a valid state.

If the optional parameter, PROTECT, is specified in a MARK ON command, the device is marked as write protected. All assignments 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 SRO. A Write File mark command to any file on the device is also rejected. This option may be used for any device regardless of the state of any hardware write protect feature. It must be specified for hardware protected discs.

Possible error responses to MARK are:

FORM-ERR	Command syntax error
PARM-ERR	Operand syntax error
STAT-ERR	fd already assigned or has files assigned
BPAC-ERR	Direct-access volume I/O error encountered
DUPL-ERR	Duplicate device or volume name exists
SEQ-ERR	Attempt to mark on-line a currently on-line disc; occurs if previous volume was dismounted without being marked off.
NOFF-ERR	Attempt to mark on-line a disc which was dismounted without being marked off-line.

Display Devices

The DISPLAY DEVICES command allows the operator to determine the physical address, keys, on-line/off-line state, and the volume name (for on-line direct-access devices) of all devices in the system. The format of this command is:

```
DISPLAY DEVICES [,fd]
```

The operand fd specifies the device to which the display is routed; if omitted, the display goes to the system console.

Possible error responses to DISPLAY DEVICES are:

FORM-ERR	Command syntax error
PARM-ERR	Operand syntax error
FD-ERR	Invalid file descriptor
IO-ERR	I/O error encountered on output device or file

MAGNETIC TAPE AND FILE CONTROL COMMANDS

This set of commands allows the operator to manipulate magnetic tapes, cassettes and direct-access files from the system console. There are two general formats of these commands as follows:

```
(a)   op fd
(b)   op fd [,lu]
```

The operator op may be any one of the following:

REWIND	Rewind
RW	Rewind (alternative operator)
FRECORD	Forward-space one record
FFILE	Forward-space to filemark
BRECORD	Backspace one record
BFILE	Backspace to filemark
WFILE	Write filemark

The mnemonics REWIND and RW are both accepted for the REWIND command, for compatibility with previous operating systems and certain utility programs.

Format (a) is used for magnetic tapes and cassettes. Format (b) is used for direct-access files. The optional operand is specified if the file is assigned to more than one logical unit in the task associated with this file.

NOTE

Format (b) of these commands applies to the currently selected task, as specified by the TASK command. Refer to the section entitled Task Related Commands.

For example:

```
REW MAG1:
```

rewinds device MAG1:

```
FR PACK:SOMEFILE.OBJ,4
```

causes the file PACK;SOMEFILE.OBJ, as assigned to Logical Unit 4 of the currently-selected task, to be positioned forward one record.

Possible error responses to these commands are:

FORM-ERR	Command syntax error
PARM-ERR	Operand syntax error
IO-ERR	I/O error encountered on specified device or file or illegal or unassigned LU
ASGN-ERR	File or device could not be assigned for the reason denoted by the TYPE field
TASK-ERR	There was no currently selected task and a command using format (b) was specified

COMMAND SUBSTITUTION SYSTEM

The Command Substitution System (CSS) is an extension to the OS/32 MT Command language. It provides the user with the ability to establish files of dynamically modifiable commands which can be called from the console or other CSS files and executed in a defined sequence. In this way, complex operations can be carried out by the operator with only a small number of commands.

CSS provides more than just the ability to switch the operating system command input stream to a batch device:

—A set of logical operators are provided to control the precise sequence of commands to be obeyed.

—Parameters can be passed to a CSS file so that general sequences can be written which take on specific meaning only when the parameters are substituted.

—One CSS file can call another, in the manner of a subroutine, so that complex command sequences can be developed.

A CSS file is simply a sequential text file. It could be a deck of cards, a punched paper tape, a magnetic tape, or a disc file.

Following is an example of a simple CSS file:

```
*THIS IS A SIMPLE EXAMPLE CSS FILE
TASK .BG
ALLOCATE XXXDIX.TST
ASSIGN 2,PRT1:;*LU2-LINEPRINTER
LOAD .BG,COPY
START
$EXIT
```

Note the use of the semicolon, which allows more than one command on the same line. Note also the use of the asterisk to introduce a comment.

High Level Operator Command Package

The OS/32-MT High Level Operator Command Package is implemented as a set of CSS files. These perform a variety of the commonly used program preparation and development sequences. The package contains the following:

COMMAND	DESCRIPTION
FORT	Perform FORTRAN V compile, assembly and task establishment
FORTCLG	Perform FORTRAN V compile, assembly, task establishment, load and start
FORT6C	Perform FORTRAN VI
FORT6CA	Perform FORTRAN VI compile and assembly
FORT6CAE	Perform FORTRAN VI compile, assembly and task establishment
FORT6CLG	Perform FORTRAN VI compile, assembly, task establishment, load and start.
CAL	Perform CAL assembly and task establishment
CALCLG	Perform CAL assembly, task establishment, load and start
MAC	Perform CAL MACRO expansion, assembly and task establishment
MACCLG	Perform CAL MACRO expansion, assembly, task establishment, load and start
EDIT	Load OS Edit and Start
ESTAB	Establish an object program using the OS/32 Task Establisher (TET/32)
COPYA	Copy an ASCII file using OS Copy
COPYB	Copy a Binary File using OS Copy
COPYT	Copy an established task, resident library or overlay using OS Copy
RUN	Load and start a task
SYSGEN1,SYSGEN2	Generate a System
DEFAULT.ASN	Assign default Logical Units

For complete information refer to the *OS/32 MT High Level Operator Command Package User's Manual,* Publication Number 29-482.

Calling CSS Files

A CSS File is called and executed by naming it in a stream of commands. Any valid File Descriptor (fd) can be used, provided that there is no conflict with any of the ordinary operator commands. If the file extension is omitted and extension of "CSS" is assumed. The CSS call is the last command recognized on a command line.

In other words, the operator can cause a file of commands to be executed simply by entering the name (fd) of that CSS file. The error message, MNEM-ERR, is returned if the file does not exist as specified.

Parameters are passed to a CSS file by appending them to the call. The first parameter is separated from the file name by a space; all other parameters must be separated by commas. Null parameters are permitted.

The following are valid CSS calls:

RUN	(Calls CSS file RUN.CSS on the system volume)
CARD:	(Calls CSS file in card reader)
JUMP A,B,C	(Calls CSS file JUMP.CSS on the system volume with three parameters A,B, and C)
JUMP.CSS A,B,C	(same as previous example)
JUMP , ,C	(Calls CSS file JUMP.CSS on the system volume with three parameters, the first two of which are null)
VOLN:JUMP	(Calls CSS file JUMP.CSS on the volume VOLN)

Use of Parameters

Within a CSS file, a parameter to that file is referenced by means of the special symbol @. The first parameter is referenced by @1, the second @2, etc. A straight-forward text substitution is employed.

Thus, a CSS file RUN could consist of:

```
LOAD    @1
START   @3,@2
etc.
```

This could be called:

```
RUN  PROGRAM,NOLIST,148
```

Before each line of the CSS file is decoded, it is pre-processed, and any reference to a parameter is substituted with the corresponding text. Thus, the file RUN with the previous call is executed as:

```
LOAD PROGRAM
START   148,NOLIST
etc.
```

In general, a reference to a parameter is of the form:

@n

where n is a decimal number indicating which parameter the user is referencing. Parameters are numbered starting with 1. Parameter 0 has special meaning, defined later in this section.

A reference to a parameter is terminated by a non-decimal character.

For example, to reference parameter 12,

@12 or @12ABC or @12.EXT

are valid expressions.

Notice that this mechanism allows concatenation. For instance, if in the above file, RUN, the first command were

```
LOAD @1.TSK
```

Then only files with the extension .TSK would be presented to the loader.

Concatenation of numbers requires care. 123@1 references parameter 1, but @1123 is a reference to parameter number 1123.

A reference to a non-existent parameter is considered to be null.

The multiple @ facility enables a CSS file to access parameters of higher level files. CSS files can call each other to a maximum depth specified at system generation time. Thus, @@2 in a CSS file refers to the second parameter of the calling file.

For instance, given the CSS call,

```
CSS1  arg1,arg2
```

and assuming that in file CSS1 there is another CSS call,

```
CSS2 arg3,arg4
```

the following references may be made in CSS2:

```
@1    = arg3
@2    = arg4
@@1   = arg1
@@2   = arg2
```

If a multiple @ sequence is such that the calling level referred to is non-existent, the parameter is considered to be null.

Parameter @0 is a special parameter. It is used to reference the name of the CSS file in which it is contained. Parameter @0 is replaced, during the pre-processing of the command line, with the name of the File Descriptor in precisely the style used to call the file.

This mechanism can be used to assign the CSS file itself to a task LU. By this means the data for a program can be included in the CSS file itself. However, the program must read precisely the right number of data items or else subsequent CSS processing may fail. THIS IS ONLY VALID FOR NON DIRECT ACCESS FILES, SINCE ASSIGNING A FILE WOULD POSITION THE FILE TO THE BEGINNING AS FAR AS THE TASK WAS CONCERNED.

By simple extension, @@0 refers to the file which calls the CSS file that contains it.

Commands Executable from a CSS File

All of the commands normally available to the operator at the console can be used in a CSS file, as well as a number of commands specifically associated with the CSS facility. These additional commands are described next.

Most of the CSS commands start with the character $. If a log of commands is being kept, the $s help to emphasize where CSS has been used, but the $ has no special meaning.

Interaction of CSS with Background and Foreground

CSS is essentially a single-stream processor. It is not possible, in the general case, to write a CSS file that can fully control a complex foreground/background system. In order to control such a system, manual intervention by the console operator is often required.

It is assumed that foreground systems are controlled, under normal circumstances, by SVC 6 calls between foreground tasks. The console operator is only required to intervene in abnormal cases. The background system, however, is expected to be controlled fairly often by the operator, or by CSS files (if the background is being run in a batch-like mode).

In batch mode, CSS control is only desirable between tasks; job-control commands are not usually desired while the tasks are running. Therefore, CSS is "keyed" to the state of the background system. While a task in the background partition is in any state other than Dormant, CSS is inactive. Therefore, a START command in a CSS file should be the last command on the line if it is starting the background task. The console operator has full control over the system at these times. While the background task is Dormant, CSS is active, if a CSS file had been invoked.

The state of foreground tasks has no effect on CSS activity.

While CSS is active, the operator is still able to enter commands from the system console. The execution of these commands may be delayed while a CSS command is being read; however, this delay should not be excessive under normal circumstances. If the operator has invoked an interactive device, however, such as TTY or CRT as a CSS file, the delay may become great. Therefore, CSS should not be invoked from such devices unless the operator is willing to relinquish a great measure of system control.

If a CSS file is active, any attempt to call another CSS file from the system console is rejected.

CSS files are permitted to affect foreground tasks; a TASK command read from a CSS file establishes the currently-selected CSS task. Commands from the console are not affected by TASK commands read from CSS, and CSS is not affected by TASK commands read from the console. All task related commands (see the section entitled Task Related Commands) and CSS return code testing encountered in a CSS file affect the currently-selected CSS task.

When a CSS file is activated from the console, the currently selected CSS task is set equal to the currently-selected task. If the currently-selected CSS task is deleted from the system, any subsequent task-related or CSS return code testing commands are rejected with a task error (TASK-ERR). Also, if the currently-selected CSS task is cancelled, the active CSS File is aborted as if a $CLEAR command had been executed.

$EXIT and $CLEAR

These two commands are provided for exiting from CSS files. $EXIT causes control to return to where it was when the CSS file was called. Control returns either to the console or to a higher CSS file. A $EXIT must terminate each CSS file called.

$CLEAR causes unconditional return of control to the console. This command may be entered at the system console at any time to abort an active CSS file.

$JOB and $TERMJOB

These commands delimit a CSS job. The CSS job concept is a defensive mechanism which protects one user from the errors of a previous user. A CSS job consists of all the operator commands and tasks loaded and started between a $JOB and $TERMJOB pair. The $JOB command delimits the start of a CSS job. The return code of the background partition is reset to zero.

$TERMJOB delimits the end of a CSS job. Most errors encountered in executing operator commands in a CSS job cause the remaining statements in the CSS file to be skipped until a $TERMJOB is encountered. If the $TERMJOB is omitted, errors may cause a subsequent $JOB statement to be skipped. Some errors cause control to be returned to the console.

By separating independent users into CSS jobs delimited by $JOB and $TERMJOB statements they can be safely batched in a CSS file, eliminating the chance that errors in one job affect another. A $JOB and $TERMJOB must be on the same CSS level.

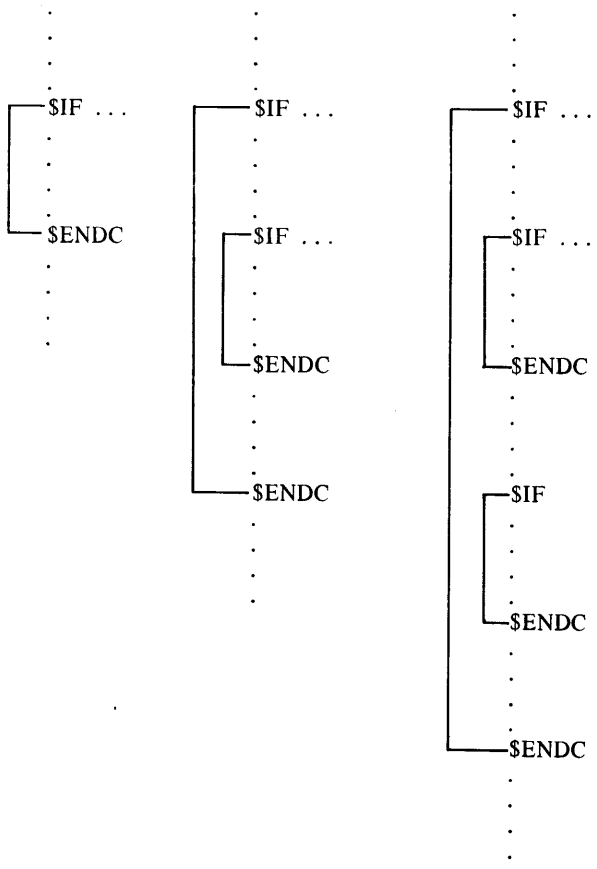
Logical Operators

There are ten logical operators available. They all start with the three characters $IF and allow one argument (e.g., $IFE 255, $IFX B.CSS, $IFNULL @1).

Each logical statement establishes a condition which is tested by the CSS processor. If the result of this test is true , then commands up to a corresponding $ENDC command are executed. If the test gives false these same commands are skipped.

The $ENDC command delimits the range of a logical operator; however, nesting is permitted, so each $IF must have corresponding $ENDC.

In the following examples, the ranges of the various conditionals are indicated by brackets.



There is no practical restriction on the depth of nesting.

The logical operators fall into three categories: Return Code testing, file existence testing, and parameter existence testing.

Return Code Testing

The Return Code is a halfword quantity maintained for each partition by the system (also see the description of SVC 3 in Chapter 3).

It is set in any of the following ways:

SET CODE n –	This command, which can be included in a CSS file or entered at the console, sets the Return Code for the currently-selected CSS task to n.
$JOB –	As part of its start job function, this command resets the Return Code for the currently-selected CSS task to zero.
Command Error –	Any command error causes the CSS mechanism to skip to $TERMJOB (assuming that a $JOB has been executed; if not, control returns to the console). To indicate that the skip has taken place, the Return Code for the currently-selected CSS task is set to 255.
$SKIP –	This command has the same effect as a command error.
EOT (SVC 3,n) –	When any task terminates by executing the EOT program command (SVC 3,n) the Return Code for that task is set to n.
CANCEL –	When a task is CANCELed, the Return Code for that task is set to 255.

There are six commands available for testing the return code of the currently selected CSS task:

$IFE	n	Test Return Code equal to n
$IFNE	n	Test Return Code not equal to n
$IFL	n	Test Return Code less than n
$IFNL	n	Test Return Code not less than n
$IFG	n	Test Return Code greater than n
$IFNG	n	Test Return Code not greater than n

In all cases if the test gives 'false', CSS skips commands until the corresponding $ENDC. If such skipping attempts to skip beyond a $TERMJOB or End of File, a command error is given.

NOTE

The return code can only be checked if the terminating task is memory resident; otherwise, the return code is always zero.

In order to run a multi-step operation in the background, where steps depend on the results of the previous steps, the background partition should be made resident. It is only meaningful to test the return code of a resident task.

File Existence Testing

There are two commands concerned with the existence of files:

$IFX	fd	Test fd for existence
$IFNX	fd	Test fd for nonexistence

Again, if the test gives 'false', CSS skips to the corresponding $ENDC. The restrictions on skipping also apply.

Parameter Existence Testing

There are two commands concerned with the existence of parameters:

$IFNULL	@n	Test @n null
$IFNNULL	@n	Test @n not null

Again, if the test gives 'false', CSS skips to the corresponding $ENDC, with the same restrictions.

The use of the multiple @ notation to test for the existence of higher level parameters is permitted.

In addition, a combination of parameters can be simultaneously tested.

For example,

```
$IFNULL @1@2@3
```

In effect, this tests the logical AND of @1,@2, and @3 for nullity. If any of the three is present, then the test results in 'false'.

Listing Directives

Two commands are provided to control the listing of CSS files as they are executed:

$COPY and $NOCOPY.

$COPY causes subsequent command lines to be listed, in their expanded form after parameter substitution. The listing takes place on the console or the log device, according to the options selected in a previous SET LOG command.

$NOCOPY switches off the listing (The $NOCOPY statement is logged.) The default is $NOCOPY.

CSS File Construction

There are two command pairs provided for construction of CSS files: BUILD, ENDB, and $BUILD, $ENDB.

BUILD and ENDB

The BUILD command causes succeeding lines to be copied to a specified file, up to but excluding the corresponding ENDB command. The format of the BUILD command is:

```
BUILD   fd
```

where fd is the new CSS file. If fd does not already exist, it is created. (A chained file is allocated with a logical record length equal to the SYSGENed command buffer length and keys of 0000; FD-ERR is issued if chained file support is not in the system.)

BUILD can be issued from the console or from within a CSS file. No nesting of BUILD commands is possible. The processing of BUILD ends when the first ENDB command is encountered, so any attempt to nest BUILD commands results in a corrupt CSS file being constructed.

The BUILD command must be the last command on its input line. Any further information on the line is treated as comment and is not copied to the new CSS file.

The ENDB command must be the only command on a line, and must occupy the first four character positions on the line. Any further information on the line is treated as comment and is ignored.

A $BUILD . . . . $ENDB sequence can be nested inside a BUILD . . . . ENDB pair.

$BUILD and $ENDB

These commands operate in a similar manner to BUILD and ENDB, except that before each line is copied to the CSS file, the CSS pre-processor substitutes any parameters in the line. It follows that $BUILD is only sensibly used from within a CSS file so that parameters can be passed to it. For instance, $BUILD is normally used to build a command file for a program (e.g., TET/32) from within a CSS file. The $BUILD command has the following format:

```
$BUILD   fd
```

where fd is the new CSS file. If fd does not already exist, it is created (as with BUILD).

As with BUILD, no nesting of $BUILD is possible. A corrupt CSS file results if the attempt is made.

$BUILD must be the last command on its input line, any further information is treated as comment and ignored.

$ENDB must be the only command on its input line, and it must occupy the first five character positions on the line. Any further information is treated as comment and ignored.

A BUILD . . . . ENDB sequence can be nested within a $BUILD . . . . $ENDB pair.

$JOB	Start next job, reset Return Code
$TERMJOB	End of Job, any error skip in last job stops at this command with Return Code = 255, otherwise, Return Code is defined by the job itself.
$EXIT	Exit from CSS file.
$CLEAR	Return control to console.
SET CODE n	Set Return Code to n.
$IFE n	If Return Code equals n, continue executing commands, otherwise skip to corresponding $ENDC.
$IFNE n	If Return Code not equal to n, continue executing commands, otherwise skip to corresponding $ENDC.
$IFL n	If Return Code less than n, continue executing commands, otherwise skip to corresponding $ENDC.
$IFNL n	If Return Code not less than n, continue executing commands, otherwise skip to corresponding $ENDC.
$IFG n	If Return Code greater than n, continue executing commands, otherwise skip to corresponding $ENDC.
$IFNG n	If Return Code not greater than n, continue executing commands, otherwise skip to corresponding $ENDC.
$IFX fd	If fd exists, continue executing commands, otherwise, skip to corresponding $ENDC.
$IFNX fd	If fd does not exist, continue executing commands, otherwise skip to corresponding $ENDC.
$IFNULL @n	If parameter does not exist, continue executing commands, otherwise skip to corresponding $ENDC.
$IFNNULL @n	If parameter exists, continue executing commands, otherwise skip to corresponding $ENDC.
$ENDC	Delimits above conditionals.
$COPY	Switch on listing.
$NOCOPY	Switch off listing
$BUILD fd	Construct CSS file with parameter substitution.
$ENDB	End of $BUILD.
BUILD fd	Construct CSS file without parameter substitution.
ENDB	End of BUILD
$SKIP	Skip to $TERMJOB

CSS Error Conditions

ERROR	MESSAGE	ACTION TAKEN
Task active	SEQ-ERR	Returns control to console
Command not recognized	MNEM-ERR	Skips to $TERMJOB
Command syntax error	FORM-ERR PARM-ERR	Skips to $TERMJOB
Second $JOB found	JOBS-ERR	Returns control to console
End of File found while skipping to $ENDC	IO-ERR	Skips to $TERMJOB
$TERMJOB found while skipping to $ENDC within a job	IO-ERR	Sets return code to 255 and ends job. (This is only detected if the conditional that caused the skip was also inside the job; i.e., a skip to $ENDC can skip over a complete job).
End of file found before ($) ENDB while ($) BUILDing a file	IO-ERR	Skips to $TERMJOB
Not enough space to build an fd, or chained file support not in system	FD-ERR	Skips to $TERMJOB
Expanded command line exceeds CSS buffer	BUFF-ERR	Skips to $TERMJOB
Too many CSS levels required	LVL-ERR	Returns control to console
Currently selected CSS task not in system	TASK-ERR	Skips to $TERMJOB

NOTE

Skips to $TERMJOB – This action only occurs if the error is detected within a CSS job. The job is aborted and the next command obeyed is the first command after the $TERMJOB, at which point the return code is 255. If the error occurs outside a job, control is returned to the console.

APPENDIX 1
SUPERVISOR CALLS

Supervisor Call (SVC) instructions provide the Program Interface to OS/32 MT. The general form of an SVC instruction is:

SVC n,P

where n represents the type of SVC and P represents an address of a parameter block.

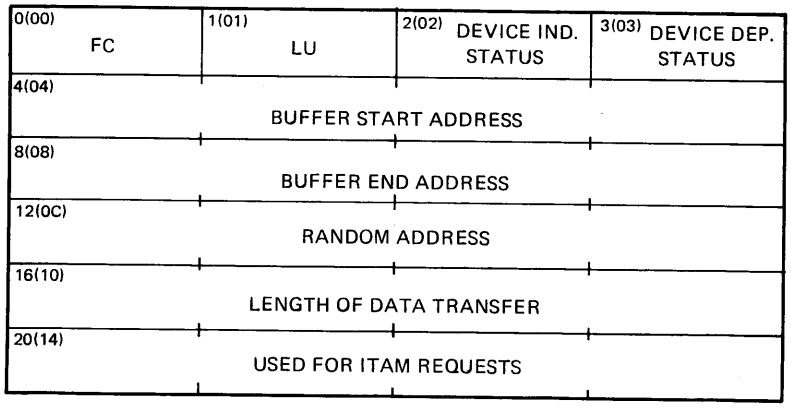
Parameter blocks must be aligned on fullword boundaries.

Unrecognized SVC types generate an illegal SVC interrupt.

Address parameters in parameter blocks must reference locations in the user tasks' memory allocation.

SVC 1 - INPUT/OUTPUT OPERATIONS

SVC 1 - PARAMETER BLOCK




```
ALIGN 4
DB X'FUNCTION CODE'
DB X'LOGICAL UNIT'
DS 2
DC A(START)
DC A(END)
DC RANDOM
DS 4
DS 4
```

Buffers must be aligned on fullword boundaries.

The full parameter block is always required, although the START ADDRESS, END ADDRESS, RANDOM ADDRESS and LENGTH fields are not necessarily used by the individual drivers in all cases.

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 00 - Wait only or Test I/O Complete
3	. . .x	ASCII/BINARY bit. This bit indicates the type of formatting requested. 0 - indicates ASCII formatting 1 - indicates binary formatting If bit 7 is set, this bit is ignored.
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 connection 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. **NOTE** If this is the only function code bit set, the request is interpreted as TEST I/O COMPLETE.
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).

SVC 1 COMMAND FUNCTION CODE

Bit	Alignment	Meaning
0	x.	This bit must be set to indicate a command function request
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

Bits 1-7 reset (Function Code X'80') implies HALT I/O

The ASCII/Binary, Formatted/Image and Sequential/Random modifiers for data transfer functions have general meanings, but specific protocol interpretations are device dependent. ASCII/Binary modifier only has meaning if Formatted is specified. Interpretation of command functions is device dependent, although the illustrated commands are conventional for devices that can support them.

INTERPRETATION OF SVC 1 DEVICE INDEPENDENT STATUS BYTE

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'

Interpretation of SVC 1 Device Dependent Status Byte

X'dn'	device number if no dependent status available
X'82'	I/O terminated by time-out
X'81'	I/O terminated by Halt I/O

Condition code is always zero except for unconditional proceed calls.

The Status fields are set to zero on normal termination.

SVC 2 - SYSTEM UTILITY SERVICES

```
        SVC     2,PARBLK        UTILITY CALL
        ...
PARBLK  ALIGN   4
        DB      OPTIONS, CODE   OPTIONS and FUNCTION CODE
        ...                     PARAMETERS
```

Function codes specify type of SVC 2 call. Unrecognized function codes are treated as illegal SVC instructions.

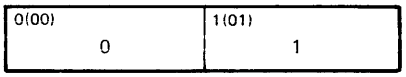
Option byte contains modifier bits as appropriate for function code, otherwise must be zero.

SVC 2 functions recognized by OS/32 MT are:

Code Number	
1	PAUSE
2	GET STORAGE
3	RELEASE STORAGE
4	SET STATUS
5	FETCH POINTER
6	UNPACK BINARY NUMBER
7	LOG MESSAGE
8	INTERROGATE CLOCK
9	FETCH DATE
10	TIME WAIT
11	INTERVAL WAIT
15	PACK NUMERIC DATA
16	PACK FILE DESCRIPTOR
17	SCAN MNEMONIC TABLE
18	MOVE ASCII CHARACTERS
19	PEEK
20	EXPAND ALLOCATION
21	CONTRACT ALLOCATION
23	TIMER MANAGEMENT

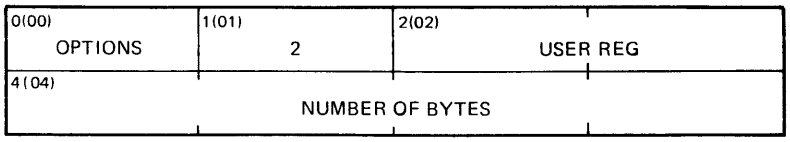
SVC 2 parameter blocks are illustrated in the following subsections.

Code 1 - Pause




```
ALIGN 4
DB    0,1
```

Code 2 - Get Storage




```
ALIGN 4
DB    OPT,2
DC    H'REG'
DC    F'SIZE'
```

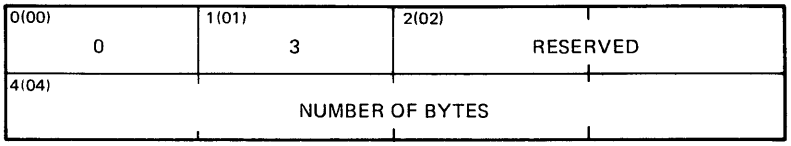
OPT = X'00' GET specified number of bytes
OPT = X'80' GET all allocated storage between UTOP and CTOP
REG is returned containing starting address of storage obtained or zero if request could not be honored.

SIZE specifies number of bytes to obtain for Options X'00'. The size field is returned set to the number of bytes obtained for Option X'80'.

The Condition Code is set:

CC = 0, if storage obtained
CC = 4, if request not honored

Code 3 - Release Storage



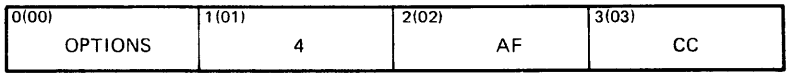

```
ALIGN 4
DB 0,3
DC H'0'
DCF F'SIZE'
```

There are no options. SIZE specifies number of bytes to release.

The Condition Code is Set:

CC = 0, if storage released
CC = 4, if request not honored

Code 4 - Set Status



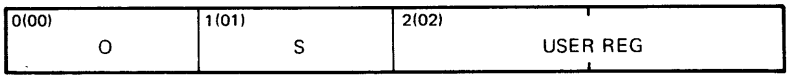

```
ALIGN 4
DB OPT,4,AF,CC
```

OPT = X'00' Modify Status and Condition Code
OPT = X'80' Modify Condition Code Only

AF = X'00' Disable Arithmetic Faults
AF = X'10' Enable Arithmetic Faults

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

Code 5 - Fetch Pointer

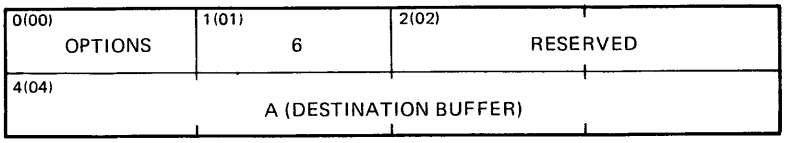



```
ALIGN 4
DB    0,5
DC    H'REG'
```

There are no options.

REG is returned pointing to a table of system parameters, of which the first three fullwords are CTOP, UTOP, and UBOT.

Code 6 - Unpack Binary Number



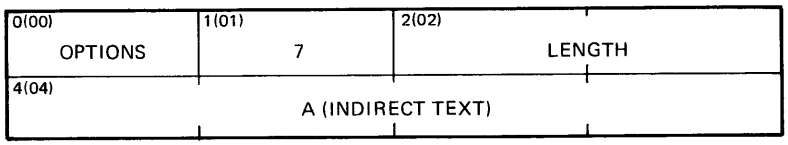

```
ALIGN  4
DB     OPT,6
DC     H'0'
DCF    A(BUFFER)
```

The binary number contained in user register zero is translated into ASCII hexadecimal or decimal format. The converted data is stored in buffer pointed to by A(DESTINATION BUFFER).

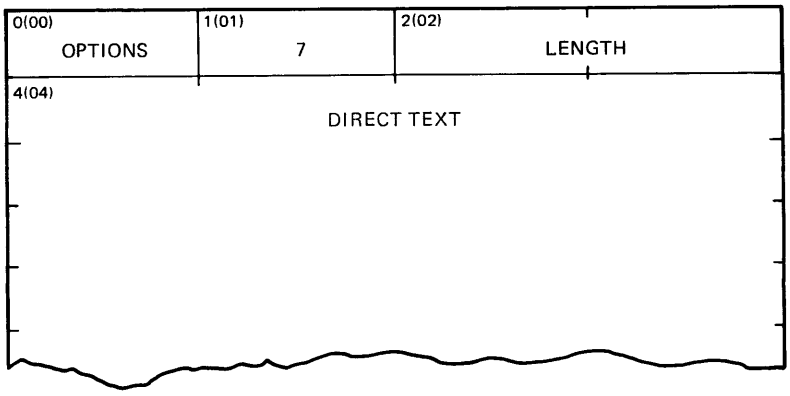
OPT = X'00' + N Convert to Hexadecimal
OPT = X'80' + N Convert to Decimal
OPT = X'C0' + N Convert to Decimal, suppress leading zero
OPT = X'40' + N Convert to Hexadecimal, suppress leading zero

Where 'N' is the length of the DESTINATION BUFFER in bytes. If 'N' = 0, a value of 4 is assumed.

Code 7 - Log Message




```
ALIGN  4
DB     OPT,7
DC     H'LENGTH'
DC     A(TEXT)
```



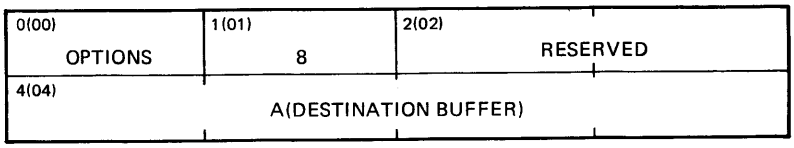

```
ALIGN  4
DB     OPT,7
DC     H'LENGTH'
DC     C'DIRECT TEXT'
```

OPT = X'40'	Indirect text, ASCII formatted message
OPT = X'C0'	Indirect text, ASCII image message
OPT = X'00'	Direct text, ASCII formatted message
OPT = X'80'	Direct text, ASCII image message

Indirect text may start on any byte boundary.

A Log Message request is treated as I/O Proceed.

Code 8 - Interrogate Clock

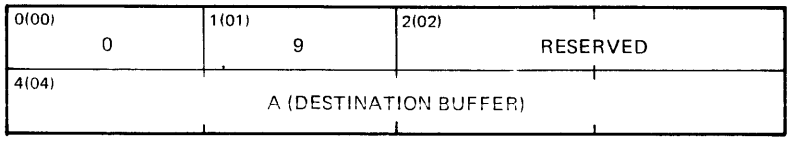



```
ALIGN  4
DB     OPT,8
DC     H'0'
DCF    A (BUFFER)
```

The current time of day is stored in the DESTINATION BUFFER in ASCII or binary format.

OPT = X'00'	ASCII format, destination buffer must be 8 bytes long and can begin on a byte boundary.
OPT = X'80'	Binary format, destination buffer must be 4 bytes long and located on a fullword boundary.

Code 9 - Fetch Date

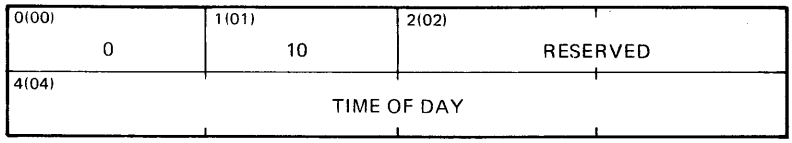



```
ALIGN 4
DB    0,9
DC    H'0'
DCF   A (BUFFER)
```

The current date is stored in the DESTINATION BUFFER. The receiving buffer must be eight bytes long and can begin on a byte boundary. The date is returned in the following format:

mm/dd/yy or dd/mm/yy controlled by SYSGEN option

Code 10 - Time of Day Wait

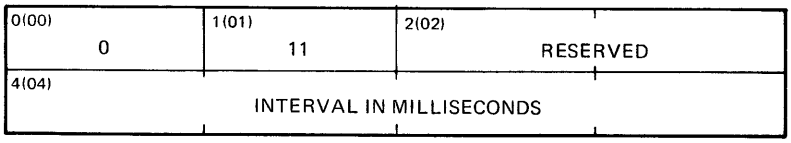



```
ALIGN 4
DB    0,10
DC    H'0'
DCF   F'TIME'
```

The calling task is suspended until a specific time of day. The Time of Day is specified in binary as seconds since midnight of the current day. The time of day field is masked to a 28-bit binary (values greater than 86399 refer to future days).

If there is insufficient system space for a timer queue entry, the wait does not take place and the condition code 'V' bit is set (CC = 4).

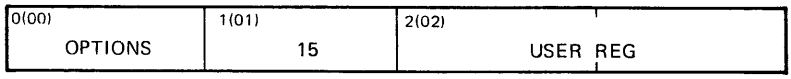
Code 11 - Interval Wait




```
ALIGN 4
DB  0,11
DC  H'0'
DCF F'TIME'
```

The calling task is suspended for a given interval of time. The interval is specified in milliseconds from the time the call is executed and it is masked to a 28-bit binary value. If there is insufficient system space for a timer queue entry, the wait does not take place and the condition code 'V' bit is set (CC = 4).

Code 15 - Pack Numeric Data




```
ALIGN 4
DB  OPT,15
DC  H'REG'
```

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

The converted ASCII representation is returned as a binary number in the user register zero; it is set to zero if no character could be processed.

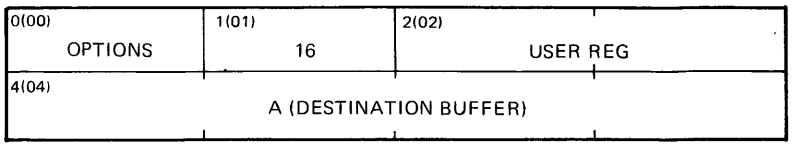
REG points to the ASCII string to be converted. It is returned pointing to the first byte in the string that could not be converted (non-numeric if decimal is specified other than 0-9 and A-F if hexadecimal is specified), unless Register 0 is specified.

The Condition Code is set to reflect the number of characters processed:

CC = 0, if characters are processed with no overflow
CC = 1, if no characters are processed
CC = 4, if conversion overflows Register 0

On overflow, for hexadecimal the least significant eight characters are returned in Register 0; for decimal the returned number is modulo 2^{31}.

Code 16 - Pack File Descriptor




```
ALIGN 4
DB   OPT,16
DC   H'REG'
DCF  A(BUFFER)
```

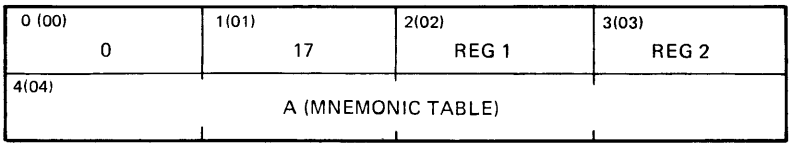
OPT = X'00' Default system volume
OPT = X'40' Default system volume, skip leading blanks
OPT = X'80' No default
OPT = X'C0' No default, skip leading blanks

The File Descriptor ASCII string, expressed in command syntax, pointed to by the contents of REG, is placed in the 16-byte destination buffer in SVC 7 format. REG is returned pointing to the first character after the end of the File Descriptor.

The Condition Code is set:

CC = 0, if full File Descriptor is processed
CC = 1, if no volume name was specified (even if default is taken)
CC = 4, if there is a syntax error
CC = 8, if no extension name was specified (defaults to blanks).
CC = 9, if no extension or volume name was specified.

Code 17 - Scan Mnemonic Table




```
ALIGN 4
DB   0,17,R1,R2
DCF  A (TABLE)
```

REG1 contains a pointer to the input 7-bit ASCII string to be matched against entries in a mnemonic table. REG1 is returned pointing to the byte in the string following the match.

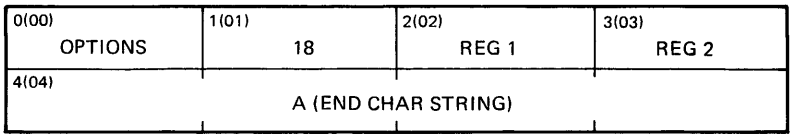
REG2 is returned with the index number of the matched entry in the table (-1 if no match, 0 if first word, etc).

The Condition Code is set:

CC = 0, if match is found
CC = 1, if no match is found

Entries in the mnemonic table are separated by a null byte (X'00'); two null bytes end the table. Each character required to be matched in an entry is indicated by its high order bit (Bit 0) being set.

Code 18 - Move ASCII Characters




```
ALIGN 4
DB   OPT,18,R1,R2
DCF  A (STRING)
```

OPT = X'00' + N No Ending Character
OPT = X'80' + N Use Ending Characters

Where N is the maximum number of characters to be moved.

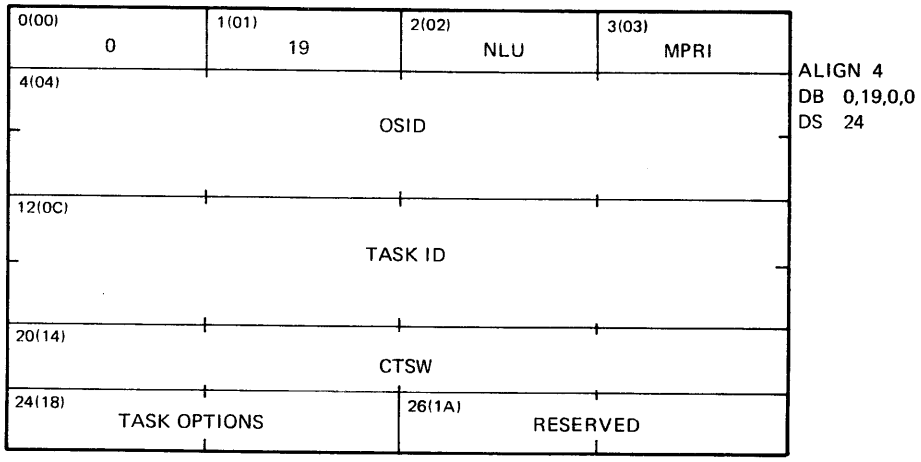
REG1 contains a pointer to the input string. REG2 contains a pointer to the receiving area. Both registers are returned pointing to the byte following the last byte moved.

A(END CHAR STRING) points to string of ASCII characters, any one of which halts the move and is not included in the move; the first byte contains the number of ending characters.

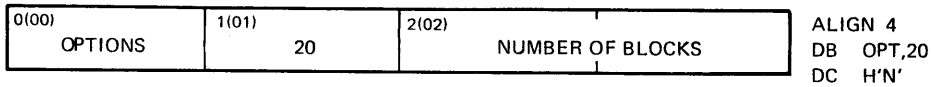
The Condition Code is set:

CC = 4, if no ending character is found with OPT = X'80'.
CC = 0, otherwise.

Code 19 - Peek



Code 20 - Expand Allocation



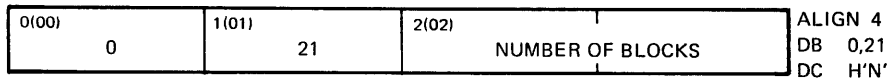
This call is provided for compatibility with earlier 32-Bit operating systems.

Options recognized are:

OPT = X'00'
OPT = X'80'

With options, X'00', the Condition Code is set to 4. With options X'80', the N field is set to zero and the Condition Code is set to 1.

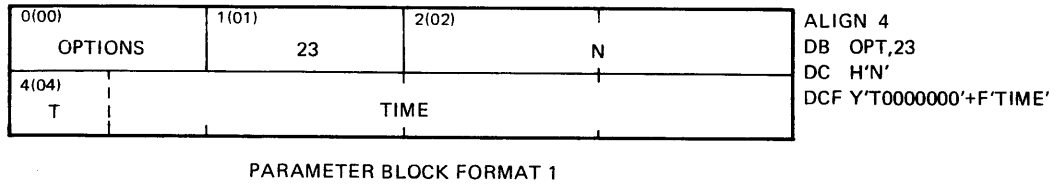
Code 21 - Contract Allocation



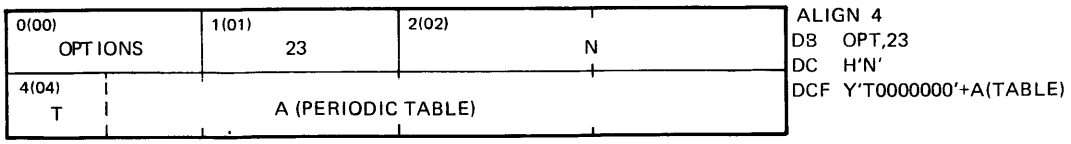
This call is provided for compatibility with earlier 32-Bit operating systems. OS/32 MT takes no action upon receiving this call, and returns immediately to the user.

The Condition Code is set to 0.

Code 23 - Timer Management



PARAMETER BLOCK FORMAT 1



PARAMETER BLOCK FORMAT 2

Format 1 causes the calling task to either wait for a single interval or time of day, or have an item added to its task queue upon completion of an interval or reaching a time of day (additions to the Task Queue cause a trap if enabled). It is also used to set up periodic time intervals. Bits 0-3 (T) of the TIME field indicate the type of interval desired:

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

Bits 4-31 of the TIME field contain the Time of Day to be waited on (Bits 0-3 = 0000) or length of interval in milliseconds (Bits 0-3 = 0001).

Options recognized are:

X'00'	Add to Queue on completion of a single interval
X'80'	Wait until completion of interval
X'20'	Read time until next interrupt
X'10'	Cancel interval
X'40'	Request Periodic Interrupts (Parameter Block 2)

The REGISTER field specifies the register containing the parameter associated with the interval. (The REG field is ignored for the option X'80'.)

If there is insufficient system space for a timer queue entry, the wait does not take place and the condition code 'V' bit is set (CC = 4).

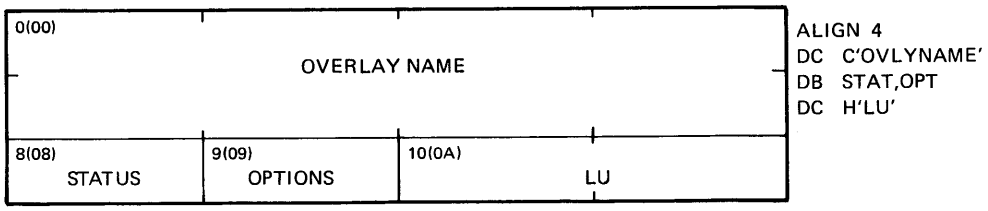
Parameter block format 2 is required to request periodic interrupts only (OPT = X'40'). Bits 4-31 of the second fullword contain the address of a table of time intervals and associated parameters. The number of pairs of entries is specified by N.

SVC 3 - END OF TASK (EOT)

There is no parameter block associated with this SVC. Instead, the resultant parameter block address of the SVC instruction is treated as a binary constant and truncated to 8 bits, and replaces the System Return Code.

SVC 5 - OVERLAY CALL

The SVC 5 parameter block format is:



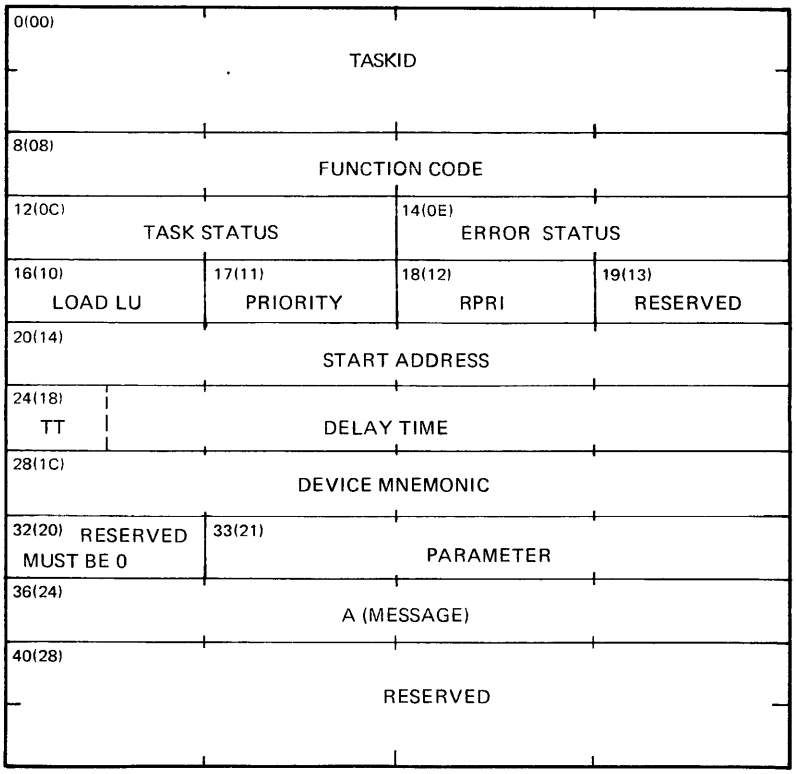
Options recognized are:

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

Status returned is:

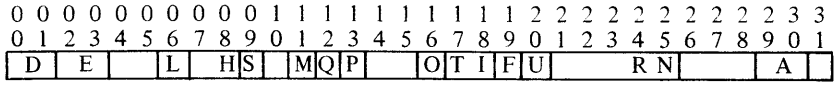
X'00'	Overlay loaded successfully
X'10'	Load failed
X'20'	Mismatch on overlay name
X'40'	Load failed, overlay would not fit in allocated memory

SVC 6 - INTERTASK COORDINATION




```
ALIGN 4
DC   C'TASKID'
DC   Y'FUNCTION CODE'
DS   4
DB   LU,PRI
DC   H'0'
DC   A(START)
DC   F'TIME'
DC   C'DEVM'
DC   Y'PARM'
DC   A(MESSAGE)
DC   0
DC   0
```

SVC 6 - FUNCTION CODES

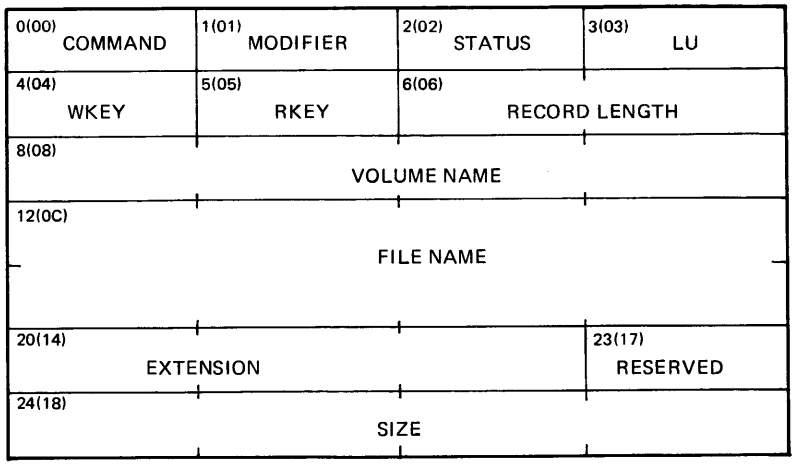


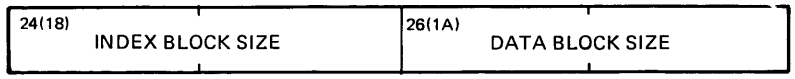
Hex Mask	Name	Meaning
C000 0000	D	DIRECTION: SELF
8000 0000	D	DIRECTION: OTHER
1000 0000	E	END TASK: CANCEL
3000 0000	E	END TASK: DELETE
0200 0000	L	LOAD
0080 0000	H	TASK RESIDENT
0040 0000	S	SUSPEND EXECUTION
0010 0000	M	SEND MESSAGE
0008 0000	Q	QUEUE PARAMETER
0004 0000	P	CHANGE PRIORITY
0000 8000	O	CONNECT
0000 4000	T	THAW
0000 2000	I	SINT
0000 1000	F	FREEZE
0000 0800	U	UNCONNECT
0000 0080	R	RELEASE SUSPEND TASK
0000 0040	N	TASK NON-RESIDENT
0000 0002	A	START: IMMEDIATE
0000 0006	A	START: DELAY

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 TASKID 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.
10(A)	A (delay)	Invalid code in TT field.
11(B)	M	Message cannot be sent.
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.
25(19)	I	Device is not SINTable.
65(41)	L	No partition with sufficient number of LUs.
66(42)	L	RTL or TCOM not present.
68(44)	L	Invalid format on Loader Information Block.
73(49)	L	No vacant partition of correct size.
128-255 (80-FF)	L	I/O error reading Load LU; error status is as returned by SVC 1.

SVC 7 - FILE HANDLING SERVICES

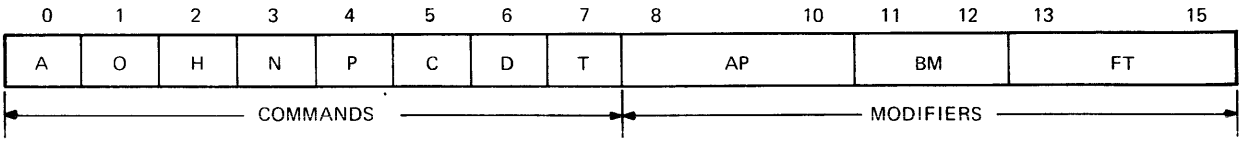




NOTE

For chained files, the fullword beginning at byte 24 contains the physical block size in sectors.

Format of the Command/Modifier halfword:



A	(Bit 0)	Allocate; requires File Type (FT) field as modifier.
O	(Bit 1)	Assign: requires Access Privilege (AP) and Buffer Management (BM) fields as modifiers.
H	(Bit 2)	Change access privileges; requires AP field as modifier.
N	(Bit 3)	Rename.
P	(Bit 4)	Reprotect.
C	(Bit 5)	Close.
D	(Bit 6)	Delete.
T	(Bit 7)	Checkpoint.

All bits zero = Fetch Attributes.

Bits 8-10 specify Access Privileges, encoded as follows:

000 = SRO	(Sharable Read Only)
001 = ERO	(Exclusive Read Only)
010 = SWO	(Sharable Write Only)
011 = EWO	(Exclusive Write Only)
100 = SRW	(Sharable Read-Write)
101 = SREW	(Sharable Read, Exclusive Write)
110 = ERSW	(Exclusive Read, Sharable Write)
111 = ERW	(Exclusive Read-Write)

Bits 11-12 specify Buffer Management, encoded as follows:

00 =	Default buffer management method.
01 =	Unbuffered Physical (default for Contiguous files).
10 =	Buffered Logical (default for Chained files and Indexed files).
11 =	Reserved, considered illegal.

Bits 13-15 specify File Type, encoded as follows:

000 =	Contiguous
001 =	Chained
010 =	Indexed
011	
. . . =	Reserved, considered illegal
110	
111 =	ITAM Buffered Terminal Manager

SVC 7 RETURN CODES

Error Code (Hexadecimal)	
00	No error
01	Illegal function
02	LU Error
03	Volume Error
04	Name Error
05	Size Error
06	Protect Error
07	Privilege Error
08	Buffer Error
09	Assignment Error
0A	Type Error
0B	File Descriptor Error
0C	TGD Assignment Error
80-FF	I/O Error

The required parameters for the command are:

ALLOCATE	FT, KEYS, LRECL, VOLN, FILENAME, EXT and SIZE
ASSIGN	AP, BM, LU, KEYS, VOLN, FILENAME, EXT; except BM, FILENAME, and EXT for direct access files only
CHANGE ACCESS PRIVILEGES	AP and LU
RENAME	LU, FILENAME, and EXT
REPROTECT	KEYS and LU
CLOSE	LU
DELETE	VOLN, FILENAME, EXT and KEYS
CHECKPOINT	LU
FETCH ATTRIBUTES	LU; returned are LRECL, VOLN, FILENAME, EXT, SIZE, device attributes in KEYS halfword and device code in MODIFIERS byte.

SVC 9 - LOAD TSW

The effective address of the SVC 9 argument specifies the location of the TASK STATUS WORD to be loaded.

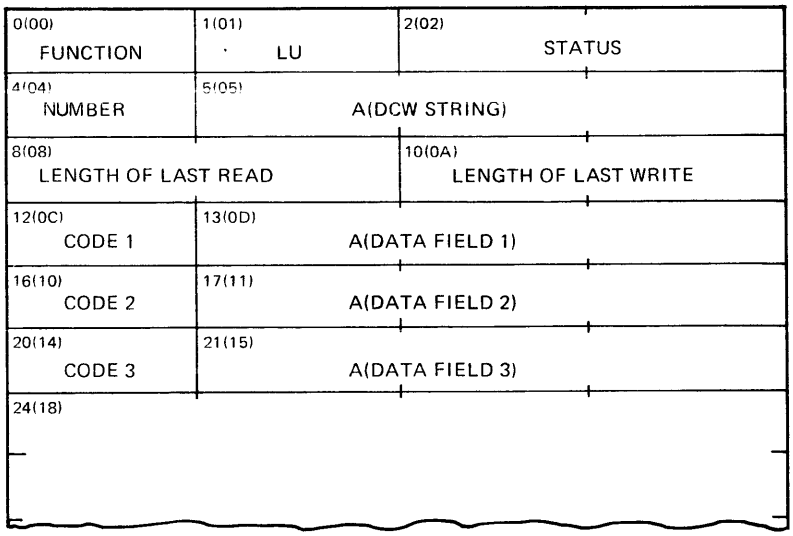
TASK STATUS WORD

Name	Bit	Mask	Meaning
W	0	8000 0000	Trap Wait
P	1	4000 0000	Power Restoration Trap Enable
A	2	2000 0000	Arithmetic Fault Trap Enable
S	3	1000 0000	SVC 14 Trap Enable
Q	4	0800 0000	Task Queue Service Trap Enable
M	5	0400 0000	Memory Access Fault Trap Enable
I	6	0200 0000	Illegal Instruction Trap Enable
D	16	0000 8000	Enable Queue Entry on Device Interrupt
T	17	0000 4000	Enable Queue Entry on Task Call
E	19	0000 1000	Enable Queue Entry on Task Message
O	21	0000 0400	Enable Queue Entry on I/O Proceed Termination
Z	22	0000 0200	Enable Queue Entry on Time-out Completion
F	23	0000 0100	Enable SVC 15 Queue Entries
cvgl	28-31		Condition Code at Time of Trap
LOC			Location Counter at Time of Trap

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

SVC 14 - USER'S SVC

The address field of SVC 14 is not interpreted by OS/32, but may be defined by the user task. If the SVC 14 Trap Enable bit in the task's current TSW is enabled, the effective address of the SVC 14 argument is stored in the SVC 14 ARGUMENT field in the task's USER DEDICATED LOCATIONS. If the SVC 14 Trap Enable bit in the task's current TSW is disabled, the SVC 14 is treated as an illegal SVC.




```
ALIGN 4
DB  FC,LU,0,0
DC  A(DCW)
DCX 0,0
DC  DC1+A(DATA1)
DC  DC2+A(DATA2)
DC  DC3+A(DATA3)
```

APPENDIX 2
COMMAND SUMMARY

```
ALLOCATE fd, CHAINED [,[lrecl[/bsize]][,keys]]

  Allocates chained file <fd> with
           <lrecl> = record length          default = 126
           <bsize> = block size             default = 1
           <keys> = protection keys         default = 00

                              Minimum abbreviation  AL filename,CH

ALLOCATE fd, CONTIGUOUS ,fsize[,keys]

  Allocates contiguous file <fd> with
          <fsize> = file size in sectors
          <keys> = protection keys         default = 00

                              Minimum abbreviation  AL filename,CO,fsize

ALLOCATE fd, INDEX [,[lrecl][/[bsize][/isize]][,keys]]

  Allocates indexed file <fd> with
          <lrecl> = record length          default = 126
          <bsize> = data block size        default = 1
          <isize> = index block size       default = 1
          <keys> = protection keys         default = 00

                              Minimum abbreviation  AL filename,IN

ALLOCATE fd,ITAM [,lrecl[/bsize][,keys]]

  Allocates ITAM terminal <fd> with
          <lrecl> = record length          default = terminal dependent
          <bsize> = block size             default = terminal dependent
          <keys> = protection keys         default = 00

                              Minimum abbreviation  AL fd,IT

ASSIGN lu,fd[,[access-priv][,keys]]

  Assigns file or device <fd> to logical unit <lu> with
          <access-priv> = access privileges default = SRW
          <keys> = protection keys         default = 00

                              Minimum abbreviation  AS lu,filename

BFILE fd[,lu]

  Backspaces file <fd> to filemark, <lu> required for
  direct access files

                              Minimum abbreviation  BF fd

BIAS  [{address or *}]

  Sets BIAS to <address> or base of partition containing currently
  selected task

                              Minimum abbreviation  BI *
```

```
BRECORD fd[,lu]

  Backspaces file <fd> one record, <lu> required for
  direct access files

                              Minimum abbreviation  BR fd

BUILD fd

  Builds text file to <fd>, if extension omitted .CSS is
  used. Subsequent input from console is written to <fd>
  until ENDB command is encountered.

                              Minimum abbreviation  BU filename

CANCEL

  Cancels current task

                              Minimum abbreviation  CA

CLOSE [lu[,lu ... ]]

  Closes specified Logical Units of currently selected task

                              Minimum abbreviation  CL lu

CLOSE ALL

  Closes all Logical Units of currently selected task

                              Minimum abbreviation  CL A

CONTINUE

  Continues  current task, if paused by operator command or
  by SVC 2 code 1.

                              Minimum abbreviation  CO

DELETE fd[,fd ... ]

  Deletes specified files.

                              Minimum abbreviation  DE fd

DISPLAY DEVICES [,fd]

  Displays on <fd> the names of the devices and direct
  access volumes in the system.
           <fd> defaults to the system console.

                              Minimum abbreviation  D D

DISPLAY FILES [,[voln:][{filename or -}[.[{ext or -}]]][,fd]]

  Displays on <fd> the files contained on volume <voln>
  with name <filename> and extension <ext>.
  -.- means all filenames, all extensions.

           <voln> defaults to the current system volume
           <fd> defaults to the system console

                              Minimum abbreviation  D F
```

```
DISPLAY ITAMTERM  [,[voln:][{filename or -}[.[{ext or -}]]][,fd]]
```

Displays ITAM terminal information on <fd>. The command syntax is interpreted as for DISPLAY FILES.

Minimum abbreviation D I

```
DISPLAY LU [,fd]
```

Displays on <fd> the current logical assignments of the currently selected task.
<fd> defaults to the system console.

Minimum abbreviation D L

```
DISPLAY MAP[/id][,fd]
```

Displays on <fd> information about partition <id>.
<id> defaults to 'whole system'
<fd> defaults to the system console.

Minimum abbreviation D M

```
DISPLAY PARAMETERS [,fd]
```

Displays on <fd> the parameters of the current task.
<fd> defaults to the system console.

Minimum abbreviation D P

```
DISPLAY TIME  [,fd]
```

Displays on <fd> the current date and time.
<fd> defaults to the current system console.

Minimum abbreviation D T

```
ENDB
```

Ends creation of text file started by BUILD command.

Minimum abbreviation ENDB

```
EXAMINE address [,n[,fd]]
```

Displays on <fd> the contents of the <n> locations, starting with <address>+BIAS.
<n> defaults to 2
<fd> defaults to the system console.

Minimum abbreviation EXA address

```
EXAMINE address1 [/address2[,fd]]
```

Displays on <fd> the contents of the bytes from <address1>+BIAS to <address2>+BIAS.
<address2> defaults to <address1>+BIAS+2.
<fd> defaults to the system console.

Minimum abbreviation EXA address

```
FFILE fd [,lu]

  Forward spaces file <fd> to filemark. <lu> is required for
  direct access files.

                              Minimum abbreviation  FF fd

FRECORD fd [,lu]

  Forward spaces file <fd> one record. <lu> is required
  for direct access files.

                              Minimum abbreviation  FR fd

LOAD taskid [,[fd][,n]]

  Loads task or library from <fd> into partiton <n> and names
  it <taskid>.
           <taskid> may also be .BG or .LIB
           <fd> defaults to TASKID.TSK for foreground tasks.
           <n> defaults to first vacant foreground partition
           large enough for task being loaded.

                              Minimum abbreviation  L taskid
                              For background  L .BG,fd
                              For library  L .LIB,fd

MARK fd,ON[,PROTECT]

  Marks direct access device <fd> on-line to the Operating
  System. PROTECT specifies device is read-only.

                              Minimum abbreviation  MA fd,ON

MARK fd,OFF

  Marks direct access device <fd> off-line to the Operating
  System.

                              Minimum abbreviation  MA fd,OF

MODIFY  address,[data][,data ... ]

  Changes location specified by <address>+BIAS to <data>.
           <data> defaults to 0.

                              Minimum abbreviation  MO address,

OPTIONS  opt [,opt ... ]

  Sets task options to <opt>. Possible values for <opt>
  are AFCONT, AFPAUSE, RESIDENT, NONRESIDENT, FLOAT, DFLOAT,
  NOFLOAT, SVCPAUSE, or SVCCONT.

                              Minimum abbreviation  OPT AFC,AFP,R,NON,F,DF,
                                                        NOF,SVCP,SVCC

PAUSE

  Pauses current task.

                              Minimum abbreviation  P
```

RENAME oldfd,newfd

Changes name of device or file from <oldfd> to<newfd>.

Minimum abbreviation REN oldfd,newfd

REPROTECT fd,keys

Changes protection keys of file or device <fd> to <keys>.

Minimum abbreviation REP fd,keys

RESET

Resets a quiescent system.

Minimum abbreviation RES

REWIND fd [,lu]

Rewinds file <fd>. <lu> is required for direct access files.

Minimum abbreviation REW fd

RW fd [,lu]

Alternative spelling for REWIND.

SEND (Up to 64 ASCII characters)

Sends characters as message to current task.

Minimum abbreviation SEN (message)

SET CODE n

Sets the return code of the CSS task to <n>.

Minimum abbreviation SE C n

SET LOG [fd[,COPY]]

Sets system log device to <fd>. COPY option allows console to receive messages also. No parameters switches log off.

Minimum abbreviation SE L or SE L fd,C

SET PARTITION id/size[,id/size ...]

Sets sizes of partitions in a quiescent system. <id> is a foreground partition number, .TCOM or .SYS
<size> is a multiple of 1KByte, in steps of .25.

Minimum abbreviation SE PA id/size

SET PRIORITY n

Sets priority of current task to <n>.

Minimum abbreviation SE PR n

SET SLICE n

Sets time slicing constant.
<n> = 0 switches off time slicing.

Minimum abbreviation SE S n

SET TIME mm/dd/yy,hh:nn:ss(or dd/mm/yy,hh:nn:ss)

Sets date and time; format defined at SYSGEN.

Minimum abbreviation SE T mm/dd/yy,hh:nn:ss

START [address][,args to prog]

Starts current task at <address>, args are passed at UBOT.
<address> defaults to transfer address set by TET/32.
<args to prog> defaults to <carriage return>.

Minimum abbreviation ST

TASK [taskid]

Sets current task. If <taskid> is omitted the system responds with the taskid of the current task.

Minimum abbreviation T

VOLUME [voln]

Sets system default volume. If <voln> is omitted the system responds with the current default volume name.

Minimum abbreviation V

WFILE fd [,lu]

Writes a filemark to <fd>. <lu> is required for direct access devices.

Minimum abbreviation WF fd

$BUILD fd

Builds a CSS text file to <fd>, until a $ENDB command is read from the command stream. Parameters are expanded as the file is built.

Minimum abbreviation $B fd

$CLEAR

Forces control back to the system console from a CSS file of any depth.

Minimum abbreviation $CL

$COPY

Switches on listing of CSS commands to the system console.

Minimum abbreviation $CO

$ENDB

Ends creation of a CSS file by a $BUILD command.

Minimum abbreviation $ENDB

$ENDC

Signifies the end of a group of conditionally obeyed commands.

Minimum abbreviation $ENDC

$EXIT

Exit from CSS file.

Minimum abbreviation $EX

$IFE n

Tests return code equal to <n>.

Minimum abbreviation $IFE n

$IFG n

Tests return code greater than <n>

Minimum abbreviation $IFG n

$IFL n

Tests return code less than <n>.

Minimum abbreviation $IFL n

$IFNE n

Tests return code not equal to <n>.

Minimum abbreviation $IFNE n

$IFNG n

Tests return code not greater than <n>.

Minimum abbreviation $IFNG n

$IFNL n

Tests return code not less than <n>.

Minimum abbreviation $IFNL n

$IFNNULL @n

Tests that parameter <@n> exists.

Minimum abbreviation $IFNN @n

$IFNULL @n

Tests that parameter <@n> does not exist.

Minimum abbreviation $IFNU @n

$IFNX fd

Tests that file <fd> does not exist.

Minimum abbreviation $IFNX fd

$IFX fd

Tests that file <fd> exists.

Minimum abbreviation $IFX fd

$JOB

Starts a CSS job.

Minimum abbreviation $J

$NOCOPY

Switches off listing of CSS commands to system console.

Minimum abbreviation $N

$SKIP

Skip to next $TERMJOB in command stream.

Minimum abbreviation $S

$TERMJOB

Terminates a CSS job.

Minimum abbreviation $T

APPENDIX 3
COMMAND ERROR SUMMARY

XXXX-ERR TYPE=YYYY POS = ZZZZ

where XXXX is error descriptor
ZZZZ represents last command item parsed
YYYY is error type for I/O, file access, and loader

The possible values of the TYPE=YYYY field are:

I/O:	LU	(illegal or unassigned LU)
	PRTY	(parity or recoverable errors)
	UNRV	(unrecoverable error)
	EOF	(end of file)
	EOM	(end of medium)
	DU	(device unavailable)
	FUNC	(invalid function for device)
file:	LU	(illegal LU)
	VOL	(no such volume/device)
	SIZE	(erroneous record length or size)
	NAME	(mismatched FILENAME.EXT)
	PROT	(mismatched protection keys)
	PRIV	(mismatched access privilege)
	BUFF	(unable to obtain FCB)
	ASGN	(LU not assigned)
	TYPE	(non-direct access device or off-line)
	FD	(illegal File Descriptor syntax)
	FUNC	(invalid function)
	IO	(I/O error during file access; second TYPE field indicates type of I/O error encountered)
SVC6	NLU	(no partition vacant with sufficient LUs)
	PRES	(specified TASKID already present in system)
	LIB	(invalid data in Loader Information Block)
	MEM	(no partition of sufficient size vacant)
	IO	(I/O error detected on specified device or file)
	NOFP	(system does not support floating point options required by task)
	SEG	(RTL or TCOM not present when trying to load a task using them)
	NMSG	(task has messages disabled)

If an I/O error occurs during execution of a file management or SVC 6 function, the following error message is output:

XXXX-ERR TYPE = IO TYPE = YYYY POS = ZZZZ

where YYYY indicates one of the above I/O error types.

The following is a list of Error Messages:

ALLO-ERR TYPE = NAME

Desired file name currently exists on the specified volume.

Corrective Action: Specify a unique file name.

ALLO-ERR TYPE = SIZE

Insufficient room on the disc to allocate the file.

Corrective Action: If possible, delete any files no longer needed; if a contiguous file is being allocated, reduce the size of the allocation.

Block size of Chained or Indexed file exceeds limit established at SYSGEN time.

Corrective Action: reduce the block size.

For a Chained or Indexed file, a zero logical record length or data block size was specified.

Corrective Action: Specify a non-zero logical record length or blocksize.

ALLO-ERR TYPE = TYPE

The volume specified is not a direct-access device.

Corrective Action: Ensure that specified volume is the disc volume name, not its device name.

ALLO-ERR TYPE = VOL

Volume name specified, or defaulted to, is not the name of any of the discs currently on-line.

Corrective Action: ensure that desired disc is on-line; if defaulting the volume name, ensure that the default volume name is correct (enter a VOLUME command).

ARG-ERR

The amount of space between CTOP and UTOP is insufficient for the Command Processor to place the arguments of the START command.

Corrective Action: A larger partition is required.

ASGN-ERR TYPE = BUFF

An attempt is being made to assign a file when there is insufficient system space available to accommodate the File Control Block (FCB).

Corrective Action: Close any files currently assigned which are no longer required or quiesce the system and increase the size of system space.

ASGN-ERR TYPE = LU

Attempt to assign to an LU that is greater than the maximum LU number specified at SYSGEN time.

Corrective Action:

1. Specify LU which is less than maximum LU number, or
2. ReSYSGEN system to specify greater maximum LU number.

ASGN-ERR TYPE = NAME

An assignment is being directed at a non-existent file.

Corrective Action: Make sure that the default volume name is correct. Check the files currently in existence on a given volume with a DISPLAY FILES command.

ASGN-ERR TYPE = PRIV

A file, which is currently assigned to a Logical Unit with a given privilege, can not be assigned to another Logical Unit because the access privileges are in conflict; e.g., an assignment of Exclusive Read/Write (ERW) is directed towards a file currently assigned for Shared Read only (SRO).

Corrective Action: Request a compatible access privilege on second assignment or change the access privileges currently associated with the file.

ASGN-ERR TYPE = PROT

The file being assigned to is unconditionally protected (Read and/or Write keys = X'FF') or the Read/Write keys specified in the assign statement do not correspond to those associated with the file.

Corrective Action: Specify the correct Read/Write keys in the assignment or reprotect the file using the REPROTECT command.

ASGN-ERR TYPE = SIZE

A chained file is being assigned and there is not enough room on the disc to allocate a physical block.

Corrective Action: Ensure sufficient space on the disc by deleting old files or reducing the block size of the file.

ASGN-ERR TYPE = TGD

Attempt to assign a TGD device that is not assignable.

ASGN-ERR TYPE = VOL

Volume name specified, or defaulted to is not the name of any of the discs currently on-line.

Corrective Action: Ensure that desired disc is on-line; if defaulting the volume name, ensure that the default volume name is correct.

BPAC-ERR

The disc is not ready or it is not readable.

Corrective Action: If disc is ready and is not write-protected, re-format using the DISC FORMATTER program, and INITIALIZE using the DISC INITIALIZE program.

BUFF-ERR

The expanded CSS line overflowed CSS buffer size.

Corrective Action: Specify larger CSS buffer length at next SYSGEN, or modify CSS statements. In the meanwhile, ensure that expanded CSS line will not overflow buffer by shortening length of unexpanded line.

CLOS-ERR TYPE = ASGN

The Logical Unit specified in the close command is not assigned.

Corrective Action: Verify the Logical Unit being closed with a D LU command.

DELE-ERR TYPE = ASGN

An attempt is being made to delete a file which is currently assigned.

Corrective Action: Close the Logical Unit assignment and repeat the delete.

NOTE

Following a system crash, the directory block for a file might incorrectly indicate that the file is currently assigned. In this case, the Disc Integrity Check must be run.

DELE-ERR TYPE = BUFF

There is insufficient available memory in System Space to perform the delete operation.

Corrective Action: Quiesce the system and enlarge system space or make more space available by closing one or more logical units.

DELE-ERR TYPE = TYPE

The volume name specified or defaulted to is not a direct-access device.

Corrective Action: Ensure that the proper disc is on-line and that the volume name is specified correctly. If using the volume name, make sure the current default volume name is correct.

DIR-ERR

A DISPLAY FILES command is directed at a disc containing one or more invalid directory entries.

Corrective Action: Run the Disc Integrity Check to verify the contents of the Disc Pack.

DUPL-ERR

When marking on a direct-access device, the volume name associated with it is an existent device or volume name.

Corrective Action: Run the Disc Initializer to change the disc volume name.

FD-ERR

The file descriptor is syntactically incorrect, or a program on the disc is being loaded and there is not enough system space for the load operation.

Corrective Action: Ensure that the volume name, file name and extension are all specified correctly.

FORM-ERR

The command line is syntactically incorrect.

Corrective Action: Verify the format of the command in Chapter 5.

I/O ERR

A device being accessed by the Command Processor is returning a non-zero I/O status.

Corrective Action:

TYPE=PRTY – Parity or other Recoverable error. Retry the operation with another unit, if possible.

TYPE=UNRV – An unrecoverable error has occurred.

TYPE=EOF,EOM – The device has reached an end of file or end of medium before completing the operation.

TYPE=DU – The device is unavailable. Ensure that the device is on-line and ready.

TYPE=FUNC – An invalid operation is being directed toward a device, e.g., attempting to write to a read-only device.

TYPE=LU – Illegal or unassigned LU. Close and reassign proper LU.

JOBS-ERR

A $JOB statement was encountered following another $JOB statement but prior to a $TERMJOB statement.

Corrective Action: The scope of $JOB must be terminated by a $TERMJOB. $JOBs may not be restarted.

LU-ERR

A Logical Unit specified in an assign statement is invalid.

Corrective Action: Verify that the desired LU is available for the assignment with a DISPLAY LU command.

LVL-ERR

The number of SYSGENed CSS levels was exceeded. At the cost of additional buffer space, specify a greater number of CSS levels the next time a SYSGEN is performed.

LOAD-ERR TYPE=IO

An I/O error was generated during the load operation.

Corrective Action: Retry the Load operation. If the same condition results, verify the status of the medium from which the task is being loaded.

LOAD-ERR TYPE = LIB

The data in the Loader Information Block is Invalid.

Corrective Action: This error most frequently occurs when an attempt is made to load a task which has not been established with the Task Establisher (TET/32).

LOAD-ERR TYPE = MEM

A load is attempted when no partition large enough is available.

Corrective Action: Quiesce the system and enlarge one partition to the minimum size specified when the task was established.

LOAD-ERR TYPE = NLU

There is no available partition with sufficient Logical Units. In OS/32 MT, all partitions support the same number of Logical Units. This error means that the number of Logical Units specified when the task was established exceeds the number of Logical Units specified at SYSGEN time.

Corrective Action: Re-establish the task, specifying a number less than or equal to MAXLU as set by the SYSGEN, or re-SYSGEN the system, increasing the MAXLU number specified to CUP/MT

LOAD-ERR TYPE = NOFP

A task requiring floating point support is being loaded and the required floating point option is not supported in the system.

Corrective Action: Re-establish the task, eliminating the floating point option if possible; if the task requires floating point support and the system contains no hardware floating point, the system must be re-SYSGENed to include Floating Point Traps (include a FLOAT S or FLOAT S,S statement in the CUP/MT input).

LOAD-ERR TYPE = PRES

The specified TASKID is already present in the system.

Corrective Action: If a duplicate TASKID was accidentally specified, merely specify another TASKID and retry the LOAD. However, if the same task is already in memory delete the first copy by first ensuring that the task is non-resident and then cancelling the original task.

LOAD-ERR TYPE = SEG

A task requiring the Run Time Library (RTL) or a Task Common Area (TCOM) is being loaded prior to establishing an RTL or TCOM partition.

Corrective Action: Ensure that RTL or TCOM is present before attempting to load a task requiring them. Ensure that global task common names were specified correctly when the task was established.

MEM-ERR

The system contains insufficient memory to support the command request. For example, a SET PARTITION command was entered and the sum of the partition sizes exceeds the available memory.

Corrective Action: Enter a DISPLAY MAP command to determine the current partition structure. If the structure resulting from the previous SET PARTITION command is unsatisfactory, repeat the SET PARTITION command.

MNEM-ERR

The entered command is unrecognizable.

Corrective Action: If a CSS call, verify that the device/volume name or file name is specified correctly. Otherwise, verify the command syntax in Chapter 5.

NODA-ERR

Direct access support does not exist in system.

Corrective Action: Reconfigure system to include direct access.

NOFP-ERR

There exists no floating-point support in the system.

NOPR-ERR

A command was entered which required more parameters than specified in the command line.

Corrective Action: Verify the syntax of the command in Chapter 5.

NULL-ERR

An attempt was made to rename the NULL device.

PARM-ERR

A command has been entered with invalid parameters.

Corrective Action: Verify the syntax of the command in Chapter 5.

PRIV-ERR

The access privilege mnemonic is syntactically incorrect.

Corrective Action: Ensure that the access privilege specified in the command is one of the following: SRO, SRW, SWO, SREW, ERO, ERW, EWO, EREW.

RENM-ERR TYPE = ASGN

The file/device cannot be assigned for ERW (required to perform the rename) because the file/device is currently assigned to at least one Logical Unit.

Corrective Action: Close any Logical Units currently assigned to the file device and repeat the RENAME command.

RENM-ERR TYPE = NAME

1. The new file name already exists in the volume directory.

   Corrective Action: Specify a unique file name in the RENAME command.

2. The new device name already exists within the Device Mnemonic Table (DMT).

   Corrective Action: Specify a unique device name in the RENAME command.

REPR-ERR TYPE = ASGN

The file/device cannot be assigned for ERW (required to carry out the reprotection) because the file device is currently assigned to at least one Logical Unit.

Corrective Action: Close any Logical Units currently assigned to the file device and repeat the REPRO command.

SEQ-ERR

A command is entered out of sequence. The following would all generate sequence errors:

- attempting to PAUSE a task when none is active
- assigning to a currently assigned Logical Unit while a task is active
- entering an OPTION command for an active task

SKIP-ERR

An attempt was made to skip beyond the end of a CSS job. The CSS job concept delimits CSS jobs with the $JOB $TERMJOB statements. The conditional CSS statements allow skipping to a conditional end statement (SENDC) if certain conditions are met. If the nesting of conditional statements is incorrect, a $TERMJOB statement can be encountered prior to terminating the scope of the conditional.

Corrective Action: Ensure that the nesting of CSS conditional statements is specified correctly.

SLOC-ERR

The start location of a task was specified below UBOT or was omitted when it was required.

Corrective Action: Specify correct starting location or reestablish the task.

SPAC-ERR

An assign on behalf of a task is refused because the system space available for task use has been exceeded.

Corrective Action: Re-establish the task with a larger maximum system space.

STAT-ERR

An attempt was made to MARK a device on/off-line while a Logical Unit was assigned to it.

Corrective Action: Close the Logical Unit assignment and retry the MARK.

SVC6-ERR TYPE = NMSG

An SVC 6 error was returned, indicating that receiving task could not receive a message trap.

Corrective Action: Refer to section entitled "Message Rings and Message Buffer Structures" for description of how to enable a task to receive messages.

TASK-ERR

A task related command was entered and there was no currently selected task.

Corrective Action: Re-enter TASK command, specifying correct TASKID.

TKID-ERR

An invalid TASKID syntax was entered on a LOAD command.

Corrective Action: Specify the correct TASKID in the LOAD command.

APPENDIX 4
SYSTEM MESSAGES

TASK RELATED ERROR MESSAGES:

taskid: TASK PAUSED

Task 'taskid' paused. Results from SVC 2 code 1 or PAUSE operator command.

taskid: END OF TASK n

Task 'taskid' has ended - n = Return Code in decimal.

taskid: ILLEGAL INSTRUCTION AT XXXXX

Illegal instruction fault detected at location XXXXX in taskid's address space

taskid: ILLEGAL SVC AT XXXXX

Illegal SVC call at location XXXXX in taskid's address space

taskid: ADDRESS FAULT IN SVC AT XXXXX

Address of SVC parameter block or an address parameter in the parameter block is outside task taskid's memory allocation, or is not aligned properly.

taskid: ARITHMETIC FAULT AT XXXXX

Arithmetic fault detected at location XXXXX in taskid's address space

taskid: MEMORY PARITY ERROR AT XXXXX

Parity machine malfunction detected at location XXXXX.

taskid: MEMORY FAULT AT XXXXX

The Task is attempting to access memory outside its allowable limits.

taskid>

SVC 1 read request to console device from task 'taskid'. Data should be entered as soon as possible to prevent blocking the console.

SYSTEM RELATED MESSAGES:

OS32MTrr-uu - Printed after system initialization
rr = release level, uu = update level

POWER RESTORE – RESET PERIPHERALS AND ENTER GO –

Power fail restore sequence – type 'GO' and carriage return to complete power recovery.

POWER RESTORE

Power fail restore sequence - No operator intervention required.

APPENDIX 5
CRASH CODES

CRASH CODE (HEX)	DESCRIPTION
1	Console device mnemonic not found in DMT.
2	Unrecoverable error on system console.
4	Not enough space in system for preSYSGENed TCOM and SYS to be allocated.
5	Illegal Error Code from SVC 6
7	Invalid VMT during MARK processing.
10	Invalid file descriptor during MARK processing.
100	Arithmetic fault not in UT/ET state. E9 contains address of current TCB. EE-EF contains PSW at time of fault.
101	Arithmetic fault not in user task. E9 contains current PSW at time of fault.
102	Illegal instruction, illegal SVC or illegal address passed in SVC not in user task. E9 contains current TCB ID, ED contains pointer to 4 bytes before pointer to message, EE-EF contains PSW at time of fault. Contents of Executive registers are saved in the save area pointed to by location X'86'.
103	Illegal instruction, illegal SVC or illegal address passed in SVC-user task not in UT/ET state. E9 contains address or user TCB, EF-EF same as for 102.
104	Memory parity error during Auto Driver Channel operation.
105	Attempt to pause system task.
106	Illegal SVC or illegal address passed in SVC with PSW not pointing after an SVC instruction. EE-EF contains PSW at time of interrupt.
107	Attempt to remove illegal TCB from ready chain. R9-TCB ID, R8-return address.
108	Attempt to remove a wait condition from, or chain, an illegal TCB ID. R8-return address, R9-TCB ID.
109	Attempt to dispatch illegal TCB ID from top of ready chain. E9-TCB ID.
10A	Attempt to dispatch ESR for illegal TCB ID. E9-TCB ID; EA-ES priority, EF-leaf address.
110	Attempt to start illegal TCB ID. U9-TCB ID. UF-start location.
111	Attempt to remove illegal wait bits from TCB. R8-return address, R9-TCB address, RD-wait bits.
112	Attempt to put illegal TCB into RS state. E9-TCB ID; EA-EB-return PSW.
113	Attempt to take illegal TCB out of RS state. U9-TCB ID.
115	Attempt to suspend illegal TCB. E8-return address; E9-return address; E9-TCB ID.
118	TCB has ready chain bit set but is not on ready chain. E8-return address, E9-TCB ID.
119	Memory fault interrupt-hardware error. EE-EF-PSW at time of fault.
120	Invalid size request to GETSYS. U3 = size.
132	Illegal SVC executed from within the system code. UF contains SVC address.
142	Memory fault in SVC executed from within system code, i.e., buffer not on fullword boundary.
152	Parity error within system code. Locations X'20'–X'26' contain the PSW at the time of the parity error.

CRASH CODE (HEX)	DESCRIPTION
162	False SYNC interrupt received from hardware.
180	Clock ESR scheduled, but no flags set.
180	Clock ESR scheduled, but no flags set.
181	Clock ESR scheduled, with PIC service flag set, but SPT.TQHD=0.
182	Clock ESR scheduled, with PIC service flag set, but there are no TQEs with evented flag set.
183	Clock ESR is removing a wait, causing something other than the system task to become the current task.
200	System Queue Service Interrupt-hardware error. EE-EF-PSW at time of fault.
201	Invalid leaf address on system queue. ED-leaf address.
202	Event for unconnected leaf. ED-leaf address.
205	Attempt to disconnect or release leaf not connected to current task. U8-return address, U9-connected TCB ID; UF-leaf address.
206	Release level 2 or greater than connection level for leaf.
207	Same as for 205 with UE-release level. Attempt to connect to invalid leaf address. U8-return address; UD-DCB/FCB pointer, UF-leaf address.
208	Attempt to modify a leaf not connected to current task. U8-return address, UE-new ESR address; UF-leaf address.
20A	Leaf queued to system mode with no task queued to leaf. EB-leaf address.
210	Entry to EVRTE not in ES state. U9-TCB ID; E0-E1-PSW at entry to EVRTE.
211	Task event count non-zero but all leaf occurrence counts zero. U9-TCB address.
212	Leaf being disconnected has occurrence count greater than TCB event count. U8-return addresses, U9-TCB address; UF-leaf address.
213	Attempt to delete a DCB from the driver time-out chain when it is not on the chain.
214	Top of ready Chain changed while propogating event for current task.
302	Attempt to walk directory link field when no directory entry currently in memory.
307	Request for FCB of invalid size
308	Attempt to release FCB with FBOT=MTOP
30A	FCB not found during release attempt
30B	Invalid DCB link field during release FCB
30C	Invalid FCB chain found during release attempt
30D	Bit map or directory leaf added to system queue
30E	Invalid save attributes
30F	Attempt to close invalid file type
320	Attempt to allocate an already allocated sector
321	Attempt to release a free sector
322	Attempt to release a free sector

APPENDIX 6
REVISION INFORMATION

This appendix provides descriptions of the new and revised features of OS/32 MT 02-01 and OS/32 MT 02-02. References are made only to manuals other than the PRM, the revision levels quoted are minimum levels. For further information in this manual, refer to the index.

Multiple Task Common/Discontiguous Memory

OS/32 MT 02-01 divides memory into local memory and global memory. Local memory corresponds to the total memory space recognized by OS/32 MT R01; it must be contiguous from physical address zero; and its upper bound is MTOP. As for previous revisions, local memory is used for System Space, an optional Reentrant Library segment, an optional local Task Common segment, all Foreground partitions and the Background partition.

Global memory is above MTOP. It need not be contiguous and is used for global Task Common segments. Global memory is defined for a system at System Generation using the CUP TCOM statement. A task uses a particular task common, local or global, by

1. Referencing it symbolically at the source level, using FORTRAN COMMON declarations or CAL COMN statements, and
2. Naming task common segments in the TCOM command to TET.

NOTE

The names used at TET time must correspond to the task common names established at System Generation.

If an attempt is made to load a task into a system which does not have the required Task Common segments, it is rejected with an error return, SVC 6 error X'42' or Operator error message: LOAD-ERR TYPE=SEG.

NOTE

If global memory is shared by more than one processor, each processor must be loaded with its own operating system.

References:

CUP - *OS/32 MT Program Configuration Manual,* Publication Number 29-389R05
TET - *OS/32 MT Task Establisher (TET/32) User's Guide,* Publication Number 29-412R03
FORTRAN - FORTRAN V Level 1 Reference Manual, Publication Number 29-360
CAL - *Common Assembler Language (CAL) User's Manual,* Publication Number 29-375

Task Common Segments Greater Than 64KB

This revision of OS/32 MT removes the limits of Task Common segments. They are now limited only by the physical constraints of the User's memory configuration.

The sizes of Task Common segments in the operating system are set at System Generation, with the CUP PARTITION command for Local Task Common, and the TCOM command for global. The Local Task Common size can be changed by the operator with a SET PARTITION command.

The sizes of Task Common segments needed by a task are set by the source level statements (FORTRAN: COMMON, CAL: COMN) which define the common blocks. TET is used to position these segments in the task address space. The TCOM command specifies the starting segmentation register for a Task Common block. For Task Commons of greater than 64KB, TET automatically allocates segmentation registers sequentially. This may cause a SEGMENT ADDRESSING ERROR message.

When a task is loaded, the loader ensures that the Task Commons in the system are large enough to accommodate the requirements of the task. If not, the load is rejected with an error return, SVC 6 error X'42' or Operator error message; LOAD-ERR TYPE=SEG.

References:

CUP - *OS/32 MT Program Configuration Manual,* Publication Number 29-389R05
TET - *OS/32 MT Task Establisher (TET/32) User's Guide,* Publication Number 29-412R03

Timer Management Facilities

The Timer Management Facilities of OS/32 MT are significantly enhanced by revision 02-01, which includes a repetitive interrupt feature. This allows application systems to interact with their environments on a periodic basis, with an economy of effort and a minimum of overhead.

The new facilities are upwards compatible with previous releases of OS/32, and tasks written to use the earlier facilities can be freely mixed with tasks that take advantage of the new features.

The Time Management SVC (2 code 23) provides the following function in OS/32 MT 02-01.

A task can place itself in time wait for a specified period.
A task can ask to be trapped (using the task queue) at a specified time.
A task can request repetitive traps (using the task queue) at defined intervals within a specified period.
A task can discover the time until the occurrence of a specified trap.
A task can cancel specified trap requests.

Time Slice Scheduling

OS/32 MT R02 provides two task scheduling algorithms for tasks at the same priority, round-robin and time slicing. Previous releases provided only round-robin.

The time slice option is under operator control. Using the SET SLICE command, the operator can:

1. Set a maximum period a task may remain current if another task is ready at the same priority.
2. Switch off time slicing.

67MB Discs

These large capacity discs are supported with the full facilities of the OS/32 MT 02-01 File Manager. The disc support is incorporated into a system at System Generation, using the device code; see Table 4-11.

Reference:

OS/32 MT Program Configuration Manual, Publication Number 29-389R05

Halt I/O

This compatible extension to the SVC 1 Input/Output facilities allows a task to abort an outstanding proceed I/O request. This is particularly useful for interactive tasks, which can now keep a terminal 'permanently live' with a proceed I/O read request while performing calculations.

The user can interrupt the calculations by typing in a command, which the task detects either by taking a trap or by using the SVC 1 Test I/O function. However, if the calculations are completed and the read is still outstanding, the task outputs results to the terminal by aborting the read with SVC 1 Halt I/O and then performing write requests as required.

Halt I/O can also be used to abort write and command function requests.

Double Precision Floating Point Support

OS/32 MT R02 provides support for Double Precision Floating Point (DPFP) in two ways:

1. Hardware Support
   The Double Precision Floating Point hardware option of the 8/32 processor is supported. The registers are saved and restored, as necessary, during scheduling, and a save area is reserved for power failure.
2. Software Emulation
   Double Precision Floating Point is emulated on systems without the hardware with DPFP Software Traps. This package of routines is called by the illegal instruction handler to emulate the DPFP hardware.

The user selects the Floating Point support for a system at System Generation, using the CUP FLOAT command. The floating point options required by a task are specified with the TET OPTIONS command, at Task Establishment, or with the Operator OPTIONS command at run time.

References:

OS/32 MT Program Configuration Manual, Publication Number 29-389R05
OS/32 MT Task Establisher (TET/32) User's Guide, Publication Number 29-412R03

Mini I/O System

The Mini I/O System of Analog and Digital conversion equipment is supported by the 02-01 revision of OS/32 MT. The system is fully integrated with the operating system and is accessed using the SVC 1 Input/Output facilities.

Mini I/O support is incorporated into a system at System Generation, using the device codes specified in Table 4-11.

Reference:

Mini I/O System User's Manual, Publication Number 29-485

Sub-Division of Source Modules

This change to the Operating System packaging only affects those users who wish to perform source system generations, or who wish to modify the OS itself. Otherwise this change is invisible to the user.

Reference:

OS/32 MT Program Configuration Manual, Publication Number 29-389R05

Multiple OS Images on Disc Files

OS/32 MT R02 allows OS images to be created in contiguous files within the standard Filing system. Such files are created using the Task Establisher (TET/32); an example is included in the TET User's Guide.

The OS/32 Direct Access Bootstrap Loader (03-074R02) is used to load and execute OSs held in these files. For upward compatibility, the Bootstrap Loader provides a method to load unnamed OS images created with earlier revisions of the 32-Bit software. The 32-Bit LSU Direct Access Loader can also be used to load OS image files.

References:

TET - *OS/32 MT Task Establisher (TET/32) User's Guide,* Publication Number 29-412R03
Bootstrap Loader - *32-Bit Family Loader Description Manual,* Publication Number 29-376R10
LSU Loader - *32-Bit LSU Direct Access Loader Instruction Manual,* Publication Number 29-472

Disc Directory Pre-Allocation

OS/32 MT 02-01 recognizes disc directories pre-allocated by the OS/32 Disc Initializer (03-081R02). Such directories are organized for optimal search time, taking into account the rotation speed of the discs.

The File Manager does not delete pre-allocated directory blocks during system operation. If the pre-allocation proves to be inadequate, further blocks are allocated in the normal fashion; however these are deleted when they become empty.

Reference:

OS/32 Disc Initializer Manual, Publication Number 29-508

Date Stamping of Disc Files

The OS/32 MT 02-01 File Manager provides date and time stamping of files. Both the date and time of allocation and date and time of last assignment are recorded in the directory. The date and time of last assignment is preset to zero on allocation.

This information is included on disc directory maps printed by the Disc Dump Utility.

Reference:

Disc Dump Utility Manual, Publication Number 29-417.

New Operator Commands

In addition to the SET SLICE command (see Time Slice Scheduling) OS/32 MT 02-01 contains the following new commands:

DISPLAY ITAMTERM
This command provides the operator with information about his ITAM configuration. The syntax of the command is similar to the DISPLAY FILES command, and the same flexibility is provided.

SEND
This command allows the operator to send text messages to foreground tasks. The tasks must have set themselves up to receive SVC 6 messages.

Modified Operator Commands

The following commands are affected by OS/32 MT 02-01.

DISPLAY MAP
Displays information about global task common segments.

EXAMINE & MODIFY
These commands protect the user against illegal addresses.

OPTIONS
Includes DFLOAT option for DPFP.

Functional Changes

Reentrant Library Segment

The reentrant library segment is mapped into segment 15 by OS/32 MT 02-01. This provides more flexibility in the use of the other segments for task common segments.

All tasks established to run under previous versions of the OS must be re-established with TET/32 R02 if they use the reentrant library or task common.

EOT Return Codes

The return code parameter to SVC 3 is truncated to 8 bits. Tasks are therefore restricted to return codes in the range 0-255. The OS forces a return code of 1000 if a task is abnormally terminated due to queue overflow on a timer trap.

OS/32 MT 02-02 Revision Information

Functional Changes

Changes to the MARK logic to prevent a disc from being MARKed ON-line unless it was properly MARKed OFF-line.

Support of 256 MB disc.

Support of FORTRAN VI in HLOC package.

New power fail source sysgen option that does not require operated intervention on a power failure.

D

E

F

G

H

I

Q

R

S

T

U

V

W

INDEX

A

B

C

PUBLICATION COMMENT FORM

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

From _____ Date _____

Title _____ Publication Title _____

Company _____ Publication Number _____

Address _____

_____

_____

FOLD FOLD

Check the appropriate item.

☐ Error Page No. _____ Drawing No. _____

☐ Addition Page No. _____ Drawing No. _____

☐ Other Page No. _____ Drawing No. _____

Explanation:

FOLD FOLD

CUT ALONG LINE

Fold and Staple

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

STAPLE

STAPLE

FOLD

FOLD

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

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

POSTAGE WILL BE PAID BY:

INTERDATA®

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

TECH PUBLICATIONS DEPT. MS 229

FOLD

FOLD

STAPLE

STAPLE