PERKIN-ELMER

OS/32 SYSTEM LEVEL

Programmer Reference Manual

48-040 F00 R02

The Perkin-Elmer Corporation, Data Systems Group, 2 Crescent Place, Oceanport, New Jersey 07757

TABLE OF CONTENTS

PREFACE ix |

CHAPTERS

1 OS/32 SUBSYSTEMS

1.1	INTRODUCTION	1-1
1.1.1	OS/32 Multiprocessing Support	1-2
1.2	SOFTWARE SUBSYSTEMS	1-3
1.2.1	Task Management Subsystem	1-5
1.2.1.1	Task Scheduling on a Model 3200MPS System	1-8
1.2.2	Job Accounting Subsystem	1-9
1.2.3	Memory Management Subsystem	1-10
1.2.4	Timer Management Subsystem	1-11
1.2.5	File Management Subsystem	1-12	
1.2.6	Input/Output (I/O) Subsystem	1-12
1.2.7	Error Recording Subsystem	1-13
1.2.8	Memory Diagnostics Subsystem	1-13
1.2.9	Loader and Segmentation Subsystem	1-13
1.2.10	Basic Data Communications Subsystem	1-14
1.2.11	Console Monitor Subsystem	1-15
1.2.12	Command Processor Subsystem	1-15
1.2.13	System Initialization Subsystem	1-16
1.2.14	Internal Interrupt Subsystem	1-16
1.2.15	Optional User Supervisor Call (SVC) Subsystem	1-17
1.2.16	Floating Point Subsystem	1-17

2 PRIVILEGED TASKS

2.1	INTRODUCTION	2-1
2.2	EXECUTIVE TASKS (E-TASKS)	2-2
2.2.1	Writing Executive Tasks (E-Tasks)	2-2	
2.2.2	OS/32 Data Structures Used by Executive Tasks (E-tasks)	2-3	
2.3	PRIVILEGED USER TASKS (U-TASKS)	2-7	
2.4	DIAGNOSTIC TASKS (D-TASKS)	2-7

CHAPTERS (Continued)

3	OS/32 SUPERVISOR CALLS (SVCs) FOR SYSTEM LEVEL PROGRAMMING	
3.1	INTRODUCTION	3-1
3.2	SVC 0: USER WRITTEN SUPERVISOR CALL (SVC)	3-3
3.3	SVC 2 CODE 0: MAKE JOURNAL ENTRIES	3-4
3.4	SVC 2 CODE 14: INTERNAL READER	3-6
3.4.1	Parameter Block for Option 0	3-6
3.4.2	Parameter Block for Option 1	3-8
3.4.3	SVC 2 Code 14 Status Codes	3-10
3.4.4	Programming Considerations	3-10
3.5	SVC 2 CODE 26: FETCH DEVICE NAME	3-12
3.6	SVC 2 CODE 27: MEMORY MANAGEMENT	3-14
3.7	SVC 6: SYSTEM TASK RELEASE	3-17
3.8	SVC 7: EXTENDED FUNCTIONS FOR PRIVILEGED TASKS	3-20
3.8.1	SVC 7: Bare Disk Assignment	3-20
3.8.2	SVC 7 Code 0: Fetch Attributes for Bare Disk Devices	3-23
3.8.3	SVC 7: Device Rename	3-25
3.8.4	SVC 7: Device Reprotect	3-26
3.8.5	SVC 7 Code X'FF80': Extended Close Function	3-28
3.9	SVC 13: AUXILIARY PROCESSING UNIT (APU) CONTROL	3-31
3.9.1	SVC 13 Code 0: Fetch Logical Processor Mapping Table (LPMT)	3-32
3.9.2	SVC 13 Code 1: Fetch Auxiliary Processing Unit (APU) Status	3-34
3.9.3	SVC 13 Code 2: Auxiliary Processing Unit (APU) Mapping	3-41
3.9.4	SVC 13 Code 3: Auxiliary Processing Unit (APU) Control	3-44
3.9.5	SVC 13 Auxiliary Processing Unit (APU) Hardware Status (SV13.APS) Field	3-50
3.9.6	SVC 13 Error Status Code (SV13.ERR) Field	3-54
3.9.7	Typical Option Coding Sequences for SVC 13 Code 2 and Code 3	3-57
3.9.7.1	Auxiliary Processing Unit (APU) Initialization and Start up	3-57
3.9.7.2	Auxiliary Processing Unit (APU) Mark On	3-57
3.9.7.3	Effective Task Scheduling on the Auxiliary Processing Unit (APU) Queue	3-58
3.9.7.4	Auxiliary Processing Unit (APU) Mark Off	3-59

CHAPTERS (Continued)

4 SUPERVISOR CALL (SVC) INTERCEPTION

4.1 INTRODUCTION 4-1

4.2 HOW SUPERVISOR CALL (SVC) INTERCEPTION WORKS 4-3

4.3 PREPARING A TASK FOR SUPERVISOR CALL (SVC) INTERCEPTION 4-4
4.3.1 Request Descriptor Block (RDB) Buffers 4-4
4.3.2 Circular List for Request Descriptor Block (RDB) Buffers 4-6 |
4.3.3 Task Event Trap 4-8 |

4.4 CREATING INTERCEPT PATHS (ICREATE) 4-8 |

4.5 HOW TO CREATE A PSEUDO DEVICE OR TASK WITH ICREATE 4-9 |

4.6 USE OF GENERIC NAMING FOR PSEUDO DEVICES AND TASKS 4-9 |

4.7 FUNCTIONAL SUMMARY OF SUPERVISOR CALL (SVC) INTERCEPTION 4-10

4.8 FULL AND MONITOR CONTROL INTERCEPT PATHS 4-11 |

4.9 HOW INTERCEPT PATHS HANDLE SUPERVISOR CALLS (SVCs) OCCURRING AT END OF TASK 4-13

4.10 TERMINATING THE INTERCEPTED SUPERVISOR CALLS (SVCs) 4-13 |

4.11 HOW TO REMOVE INTERCEPT PATHS 4-14 |

4.12 ERROR HANDLING 4-14 |

4.13 MACROS USED WITH SUPERVISOR CALL (SVC) INTERCEPTION 4-16 |
4.13.1 ICREATE Macro 4-16 |
4.13.2 IREMOVE Macro 4-22 |
4.13.3 IGET Macro 4-24
4.13.4 IPUT Macro 4-26 |
4.13.5 ICONT Macro 4-27 |
4.13.6 IPROCEED Macro 4-28 |
4.13.7 IROLL Macro 4-29 |
4.13.8 ITERM Macro 4-30 |
4.13.9 ITRAP Macro 4-32 |
4.13.10 IERRTST Macro 4-34 |
4.13.11 $RDB Macro 4-35

4.14 SAMPLE SUPERVISOR CALL (SVC) INTERCEPTION PROGRAMS 4-36

CHAPTERS (Continued)

5 OS/32 SUPPORTED INPUT/OUTPUT (I/O) DEVICES

5.1	INTRODUCTION	5-1
5.2	VERTICAL FORMS CONTROL (VFC)	5-1
5.2.1	Horizontal Tabs	5-2
5.2.2	Theory of Operation	5-2
5.3	CARD READERS	5-3
5.4	CARD READER/PUNCH DEVICES	5-4
5.5	TELETYPE (TTY) READER/PUNCH	5-6
5.6	TELETYPE (TTY) KEYBOARD/PRINTER	5-7
5.7	PAPER TAPE EQUIPMENT	5-8
5.8	LINE PRINTERS	5-9
5.9	TAPE CASSETTE	5-9
5.10	MAGNETIC TAPE	5-9
5.10.1	Standard Input/Output (I/O)	5-10
5.10.2	Gapless Input/Output (I/O)	5-11
5.11	DISK STORAGE	5-11
5.12	FLOPPY DISK	5-12
5.13	VIDEO DISPLAY UNIT (VDU) TERMINALS	5-12
5.14	8-LINE INTERRUPT MODULE	5-13
5.15	DIGITAL MULTIPLEXOR	5-13
5.16	CONVERSION EQUIPMENT	5-13
5.17	ANALOG INPUT CONTROLLER	5-14
5.18	ANALOG OUTPUT CONTROLLER	5-15
5.19	DIGITAL INPUT/OUTPUT (DIO) CONTROLLER	5-15

| 6 PROGRAMMING IN A MODEL 3200MPS SYSTEM MULTIPROCESSING
| ENVIRONMENT

6.1	INTRODUCTION	6-1
6.2	DESIGNING TASKS TO RUN ON A MULTIPROCESSING SYSTEM	6-1

6.3 ASSIGNING A TASK TO A PROCESSOR 6-2

6.4 PREPARING AN AUXILIARY PROCESSING UNIT (APU) FOR TASK EXECUTION 6-4

6.5 TASK RESCHEDULING 6-6
6.5.1 Monitoring and Preempting Auxiliary Processing Unit (APU) Task Execution 6-6
6.5.2 Verifying Task Transfer to an Auxiliary Processing Unit (APU) 6-11
6.5.3 Transferring a Task from an Auxiliary Processing Unit (APU) to the Central Processing Unit (CPU) 6-12

6.6 PREVENTING MEMORY ACCESS CONFLICTS 6-13
6.6.1 Avoiding System Deadlock 6-13

6.7 MEASURING REAL-TIME PERFORMANCE ON A MODEL 3200MPS SYSTEM 6-14

6.8 WHERE TO GO FOR MORE INFORMATION 6-17

APPENDIXES

A OS/32 SUPPORTED INPUT/OUTPUT (I/O) DEVICES A-1

B SUPPORTED VERTICAL FORMS CONTROL (VFC) CHARACTER SET B-1

FIGURES

1-1 Typical Model 3200MPS System Configuration 1-3

3-1 SVC 2 Code 0 Parameter Block Format and Coding 3-4
3-2 SVC 2 Code 14 Parameter Block Format and Coding for Option 0 3-7
3-3 SVC 2 Code 14 Parameter Block Format and Coding for Option 1 3-9
3-4 SVC 2 Code 26 Parameter Block Format and Coding 3-12
3-5 SVC 2 Code 27 Parameter Block Format and Coding 3-14
3-6 SVC 6 Release Function Parameter Block Format and Coding 3-18
3-7 SVC 7 Bare Disk Assignment Parameter Block Format and Coding 3-21
3-8 SVC 7 Code 0 Parameter Block Format and Coding 3-23

FIGURES (Continued)

3-9	SVC 7 Device Rename Parameter Block Format and Coding	3-25
3-10	SVC 7 Device Reprotect Parameter Block Format and Coding	3-27
3-11	SVC 7 Code X'FF80' Parameter Block Format and Coding	3-29
3-12	SVC 13 Code 0 Parameter Block Format and Coding	3-32
3-13	Data Buffer Format for SVC 13 Code 0	3-34
3-14	SVC 13 Code 1 Parameter Block Format and Coding	3-35
3-15	Data Buffer Format for SVC 13 Code 1	3-37
3-16	Format of APU Software Status Field Returned to Task Buffer	3-38
3-17	SVC 13 Code 2 Parameter Block Format and Coding	3-42
3-18	SVC 13 Code 3 Parameter Block Format and Coding	3-45
3-19	SVC 13 APU Control Options Field (SV13.OPT)	3-47
3-20	APU Hardware Response Byte (SV13.APS)	3-51
4-1	Request Descriptor Block	4-5
4-2	System Task Buffer List (Standard Circular List)	4-7
5-1	Random Field Format	5-14
5-2	Analog Output Data Format	5-15
6-1	Valid APU Operating States	6-5

TABLES

1-1	PERKIN-ELMER OS/32 SOFTWARE SUPPORT	1-4
2-1	OS/32 DATA STRUCTURES USED BY E-TASKS	2-4
3-1	OS/32 SVCs FOR SYSTEM LEVEL PROGRAMMING	3-1
3-2	SVC 2 CODE 14 STATUS CODES	3-10
3-3	SVC 13 FUNCTION CODES	3-31
3-4	BIT DEFINITIONS FOR APU SOFTWARE STATUS FIELD RETURNED TO TASK BUFFER	3-39
3-5	APU OPTIONS FIELD BIT DEFINITIONS	3-40
3-6	SVC 13 CODE 3 APU CONTROL OPTIONS (SV13.OPT) FIELD BIT DEFINITIONS	3-47
3-7	SVC 13 CODE 3 APU COMMANDS (SV13.DOP)	3-49
3-8	APU HARDWARE RESPONSE BYTE BIT DEFINITIONS	3-51
3-9	ERROR CODES FOR ERROR CODE BYTE OF APU HARDWARE STATUS (SV13.APS) FIELD	3-53
3-10	SVC 13 ERROR STATUS CODES (SV13.ERR)	3-55
4-1	SYSTEM MACROS FOR SVC INTERCEPTION	4-1
4-2	ERROR CODES RETURNED FOR INTERCEPT MACROS	4-15
4-3	VALID COMBINATIONS FOR SVC, MODE, AND NAME PARAMETERS	4-19

TABLES (Continued)

6-1 TIMER MACROS 6-15

6-2 ADDITIONAL INFORMATION SOURCES FOR MODEL 3200MPS SYSTEM PROGRAMMING 6-17

INDEX Ind-1

PREFACE

This manual describes privileged supervisor call (SVC) facilities and other operating system features intended for use by system programmers, system analysts, designers, engineers, and training instructors.

Chapter 1 presents an overview of the operating system and the software it supports. Chapter 2 describes the privileged task types supported by OS/32. Chapter 3 explains how to use the SVC 0, SVC 2, SVC 6, SVC 7, and SVC 13 facilities for system level programming. Chapter 4 contains a functional description of the SVC interception feature. The vertical forms control (VFC) feature is described in Chapter 5 along with other device independent and dependent features supported by OS/32. Chapter 6 describes the techniques used in writing system level control programs that take advantage of the increased throughput offered by a Perkin-Elmer Model 3200MPS System.

The R02 revision of this manual introduces enhancements to the OS/32 task manager for programming in a Model 3200MPS System multiprocessing environment. Features for efficient use of this expanded management ability include selection and assignment of tasks to specific processor units, rescheduling of tasks from one processor to another, and prevention of invalid data from concurrently running tasks. Real-time performance of individual tasks in the system is also measured.

This manual is intended for use with the OS/32 R07.1 software release or higher. Additional material that is specifically related to the Model 3200MPS System has been included. These Model 3200MPS System features are supported by the OS/32 R07.1 software release and higher. Throughout the text, these features are identified as applicable only to the Model 3200MPS System.

For further information on the contents of all Perkin-Elmer 32-bit manuals, see the 32-Bit Systems User Documentation Summary.

CHAPTER 1
OS/32 SUBSYSTEMS

1.1 INTRODUCTION

Perkin-Elmer OS/32 is a general purpose, event-driven operating system for Perkin-Elmer 32-bit computer systems. Custom versions of OS/32 are created through the use of a system generation program (SYSGEN/32) that provides parameters for tailoring OS/32 to a specific installation. The combined hardware and software capabilities of a Perkin-Elmer 32-bit computer system provide support for all phases of program and system development. OS/32 supports concurrent multiprogramming, with up to 252 user programs written in any of the supported languages. The program development facilities are designed to minimize the time and effort needed to test, debug, and integrate application programs and systems. In addition, the OS/32 command language allows complex jobs to be performed with minimum operator intervention.

OS/32 incorporates a powerful interrupt handling capability at the task level. This capability permits a task to be interrupted during its normal execution sequence by a variety of hardware and software conditions.

The OS/32 roll function and the virtual task manager (VTM) allow the memory requirements of a task running under OS/32 to exceed available task memory.

The roll function allows portions of a task to be rolled out to disk until enough memory is available for the entire task. In real-time applications, rolling is commonly used to queue low priority tasks while tasks of higher priority are active. The roll eligibility of a task is established when the task is link edited. However, a task option is provided to prevent roll of a task when necessary (e.g., when the task must be able to respond to real-time events).

VTM is a user transparent virtual memory capability that allows tasks consisting of up to 16Mb of code and data to execute in as little as 128kb of memory. This feature is provided by the OS/32 linkage editor. See the OS/32 Link Reference Manual for more information.

A basic data communications facilities package is supplied with OS/32. This package also provides support for higher level Perkin-Elmer data communications products.

The scope and power of the operating system can be extended through these Perkin-Elmer OS/32 companion products:

- Multi-Terminal Monitor (MTM)
- Reliance

MTM is a subsystem monitor that uses the subtasking capabilities of OS/32 to provide a time-sliced, interactive program development environment for up to 64 concurrent terminal users. MTM simultaneously supports both online terminal users and batch background tasks. MTM terminal users are also provided with an input/output (I/O) spooler for use with slow speed devices.

Reliance is a transaction software system, consisting of the Integrated Transaction Controller (ITC), Data Management System (DMS/32), and industry standard COBOL. ITC allocates system resources, develops screen formats, and controls terminals. DMS/32 supervises disk allocation and data access.

1.1.1 OS/32 Multiprocessing Support

OS/32 provides a transparent multiprocessing capability for use with the Perkin-Elmer Model 3200MPS System. This system consists of one central processing unit (CPU) and from one to nine auxiliary processing units (APUs). See Figure 1-1. A task can execute on an APU without any special preparation, unless it is going to take advantage of certain features specific to the multiprocessing system (e.g., APU mapping, APU control, etc.). See Chapter 3 for more information on these features.

Each processor in a Model 3200MPS System has a unique identifying number. OS/32 uses the processor numbers to define a set of logical processing units (LPUs). Each LPU is mapped to one processor number through the logical processor mapping table (LPMT). A task may execute on the processor that is mapped to the LPU assigned to the task.

If a task is mainly compute intensive, executing that task on an APU increases overall system performance. An I/O intensive task, if directed to an APU, decreases system performance since each I/O request requires the task to be transferred back to the CPU for OS/32 I/O support services.

The main performance advantage of a multiprocessing system is achieved when a problem is broken down into parts so that several tasks on several processors can work on the problem at the same time. See Chapter 6 for more information on programming from a Perkin-Elmer Model 3200MPS System environment.

6269

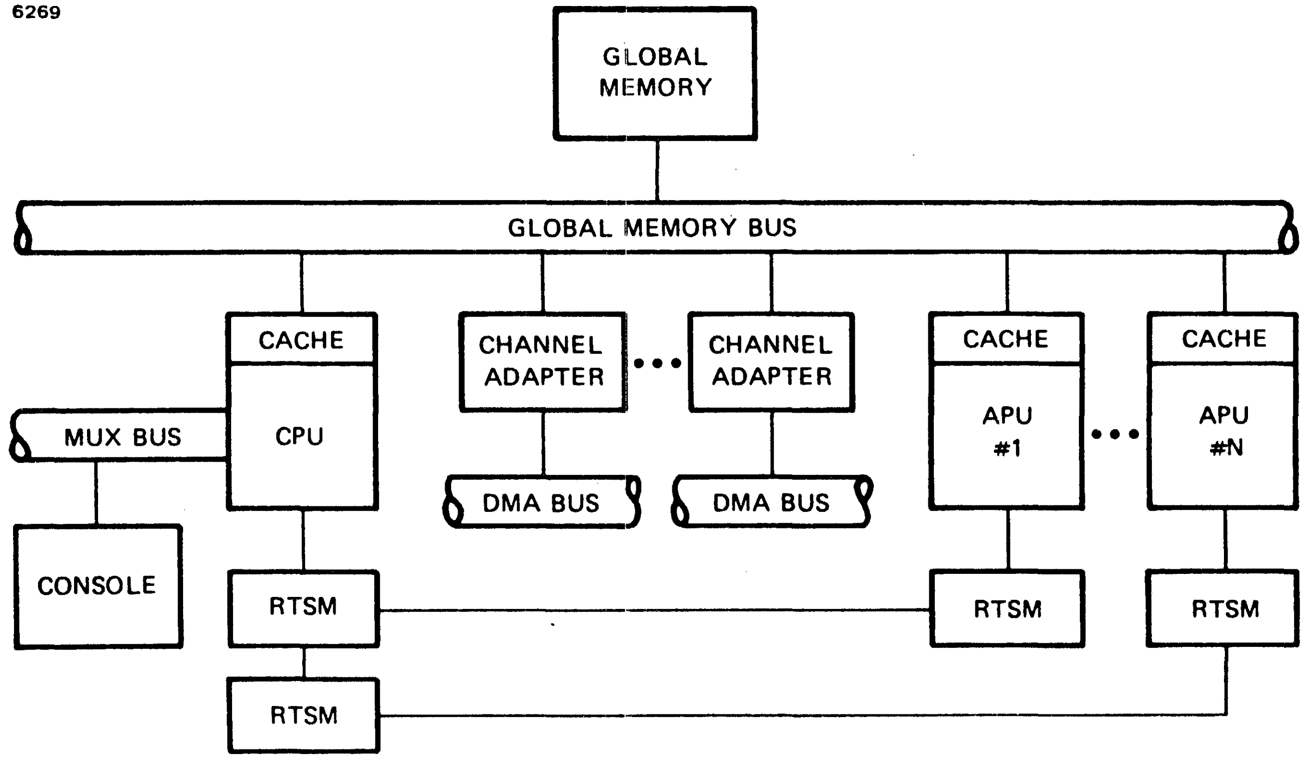


Figure 1-1 Typical Model 3200MPS System Configuration

1.2 SOFTWARE SUBSYSTEMS

OS/32 consists of the following subsystems:

- Task management
- Job accounting
- Memory management
- Timer management
- File management
- Input/Output (I/O) management
- Error recording and reporting
- Memory diagnostics
- Loader and segmentation
- Basic communications
- Console monitor
- Command processor
- System initialization
- Internal interrupt
- Optional user supervisor call (SVC 14)
- Floating point

Table 1-1 summarizes the software supported by OS/32.

TABLE 1-1 PERKIN-ELMER OS/32 SOFTWARE SUPPORT

TYPE	SOFTWARE PRODUCT	STD.	OPT.
Program Development	Task management	x	
	Job accounting	x	
	Memory management	x	
	Timer management	x	
	File management	x	
	I/O management	x	
	Error recording and reporting	x	
	Memory diagnostics	x	
	Loader and segmentation	x	
	Console monitor	x	
	Command processor	x	
	Floating point	x	
	Internal interrupt subsystem	x	
	*Integrated Transaction Controller (ITC)		x
	Writable Control Store (WCS)		x
	Multi-Terminal Monitor (MTM)		x
Program Debugging	Automatic interactive debugging system (AIDS)	x	
	DEBUG/32		x
Data Base Management	*Data management system (DMS/32)		x
Data Communications	Asynchronous data communications	x	
	Character synchronous communications	x	
	Bit synchronous communications		x
	2780/3780 RJE emulation		x
	3270 emulation		x
	HASP/32		x
Languages	Common Microcode Assembler (MICROCAL)		x
	Common Assembly Language/32 (CAL/32)	x	
	CAL Macro/32	x	
	FORTRAN VII Development (D) Compiler		x
	FORTRAN VII Global Optimizing (O) Compiler		x
	FORTRAN VII Universal Optimizing (Z) Compiler		x
	*COBOL		x
	BASIC Level II		x
	CORAL 66		x
	RPG II		x
	PASCAL		x
Utilities	Link	x	
	Edit	x	
	Text		x
	Source Updater	x	
	Copy	x	
	Library Loader	x	
	Macro Library	x	
	Sort/Merge II		x
	Patch	x	
	OS/32 Spooler	x	
	SPL/32	x	
	Fastchek	x	
	Fastback	x	
	Account Reporting Utility	x	

* ITC, COBOL, and DMS/32 comprise the Perkin-Elmer Reliance software system designed for transaction processing.

1.2.1 Task Management Subsystem

The task management subsystem allocates processor time for each of the tasks executing in an OS/32 multi-tasking environment. The task manager determines the order in which each task gains processor control on a user defined priority basis. Task priority levels range from 0 through 254 (0 being the highest priority level). Of these 255 priority levels, 10 through 249 are available for user-written tasks, while 1 through 9 and 250 through 254 are reserved for system use.

The task manager maintains four priority levels for each task:

- Maximum
- Task
- Run
- Dispatch

Maximum priority, set by Link, is the highest priority level (i.e., smallest number) that can be assigned to a task. Task priority is the priority that is currently assigned to a task. Initially, task priority is set when the task is linked, but this priority can be changed after the task is loaded. However, task priority can never be set higher than the maximum priority set by Link.

Run priority may be set dynamically to a value ranging from the task priority to task priority plus n. The value of n is based on the behavior of the task. Run priority can only be set for tasks that have dynamic time-slice/priority scheduling enabled. If dynamic scheduling is not enabled, a task's run priority is equal to its task priority. Currently, only MTM enables dynamic time-slice/priority scheduling.

A dispatched task usually has a priority level equal to its task priority, even if dynamic scheduling is enabled. Nevertheless, if a higher priority task requires specific system resources (e.g., a disk directory or bit map) that are currently controlled by a lower priority task, the dispatch priority of this lower priority task is raised to the priority of the higher priority task waiting for the resource. When a task releases control of a system resource, its dispatch priority is reset to its run priority.

Tasks competing for processor time are maintained in priority order on a task control block (TCB) queue known as the ready queue. Tasks competing for both memory and processor time are maintained in priority order on the roll-in queue. Tasks at the same priority level are serviced on a round-robin basis; i.e., tasks are added to the ready queue or roll-in queue behind tasks of the same priority.

In the absence of time-slicing, once a task gains control of the processor, it continues executing until it voluntarily relinquishes that control or is preempted by a higher priority task. A task will relinquish control of the processor to another task when:

- it is paused by the system operator;
- it is cancelled by the system operator, user, or by another task;
- a higher priority task becomes ready due to an external event, such as the completion of an outstanding I/O request;
- it executes a supervisor call (SVC) that places it in a wait, paused, or dormant state;
- it initiates I/O to a device; or
- its time-slice expires.

After the task relinquishes control of the processor, it is returned to the ready queue where its TCB is placed behind the TCBs of tasks of equal priority. This allows the other tasks on the queue to be given a turn on the processor.

To determine which task should have control of the processor, the task manager chooses the highest priority task among those on the ready queue, the roll-in queue, and any currently executing task. If a task is chosen from one of the queues, the currently executing task is placed back on the ready queue and the chosen task becomes the current task.

The task manager supports two types of time-slicing:

- System time-slice
- Dynamic time-slice

System time-slicing may be enabled by the operator to limit the execution of a task so that round-robin scheduling of priority tasks can take effect. Time-slicing allows tasks of equal priority to receive equal shares of processing time.

Dynamic time-slicing is enabled only for MTM subtasks. The dynamic time-slice is calculated as:

```
    slice = 1 + 2**m
```

Where:

```
    m = task priority - run priority + 1
```

The units of slice are line frequency clock ticks (one clock tick = 8.333 milliseconds).

Run priority is adjusted whenever a task uses up a time-slice or is removed from a wait state. When a task uses up a time-slice, its run priority is adjusted as follows:

```
    New run priority = run priority + 1 or
                       task priority + k (whichever is smaller)
```

Where:

```
    k = number of dynamically scheduled tasks or 12
        (whichever is smaller)
```

Because a task that is placed in a wait state has not used up its last assigned time-slice, the run-priority of the task when it is removed from suspension is adjusted as follows:

```
    Run priority = run priority - 1 or
                   task priority (whichever is larger)
```

The task manager also performs intra-task context switches to allow tasks to receive and handle task traps in response to synchronous and asynchronous trap-causing events. Synchronous events include task-initiated faults (e.g., arithmetic, memory access, illegal instruction, etc.) and SVC 14 traps. Asynchronous events originate outside of a task and include the task queue traps (e.g., I/O and timer completion, SVC 6 send message/data and queue parameter, etc.) and the task event traps currently associated only with SVC intercept support.

In addition to task scheduling and task trap support, the task manager handles the state of a task during execution. Task execution state is determined by the setting of the program status word (PSW). The task manager switches or exits tasks from one execution state to another.

1.2.1.1 Task Scheduling on a Model 3200MPS System

The OS/32 task manager uses four different types of queues to facilitate task scheduling on a Model 3200MPS System:

- CPU ready queue
- CPU receive queue
- APU ready queue (one per APU)
- CPU roll-in queue

When a task requests processor time on a Model 3200MPS System, the task manager adds the TCB for that task to the CPU ready queue. The task manager selects a task for execution from the queue on a strict priority basis. After selecting a task, the task manager then decides whether the task is to be executed on the CPU or on one of the APUs in the system. A task is transferred to an APU for processing only when all of the following conditions are true:

- The task is being dispatched to the user execution state. (If the task is being dispatched to a system state, it will run on the CPU.)
- The task is assigned to an LPU number greater than zero. (Each task running on the Model 3200MPS System is assigned an LPU number. An LPU number is mapped to an APU or the CPU; LPU 0 is always mapped to the CPU.)
- The task status word (TSW) does not specify CPU-override status. (If the CPU-override status bit of the TSW is set, the task is executed on the CPU.)
- The APU to which the LPU number is mapped is marked on.

When all of the above conditions are true for the highest priority task on the CPU ready queue, the task manager transfers the TCB for that task from the CPU ready queue to an APU. If the APU is waiting for the task (i.e., APU processing has been suspended until the task arrives), the TCB becomes the current TCB and execution begins immediately. If the APU is not waiting for the task, the TCB is placed on the APU ready queue.

Whenever it is not processing a task, the APU continually checks its APU ready queue. If the APU finds entries on the queue, it will execute the task at the top of the queue.

Once the APU starts a task, the task will continue to execute until it relinquishes control voluntarily (reschedules itself), encounters a fault, issues an SVC, or is returned to the CPU via an operating system request on behalf of a monitoring task, operator command, etc. The task may reschedule itself to the rear of the APU ready queue or to the CPU. In the latter case, and in the cases of a fault, or SVC or operating system request, the task is returned to the CPU receive queue. The task waits on the receive queue until the CPU places the task on the CPU ready queue.

If the task is placed on the receive queue as a result of a fault, the task is moved to the CPU ready queue. If the appropriate bits in the TSW are set, the task's TSW location is set to the address of the task trap handler. The task can then be dispatched back to the APU ready queue.

If the task is placed on the receive queue as a result of issuing an SVC, the task is moved to the CPU ready queue and executed on the CPU until SVC processing is complete. The task can then automatically move back to the APU ready queue.

Rollable tasks are moved from the roll-in queue to the CPU ready queue and are processed in the same manner as any other task running on a Model 3200MPS System. Rollable tasks may be dispatched to an APU. However, while a task is executing on an APU (or queued for execution to an APU) the task is considered nonrollable by the system.

1.2.2 Job Accounting Subsystem

The job accounting subsystem reports CPU usage and time elapsed, memory and disk space utilized, and number and length of I/O transfers by device class. The job accounting subsystem contains the:

- Data collection facility
- Account reporting utility

The data collection facility collects accounting data on all user activities and stores this information in the accounting transaction file (ATF) when the task terminates. The account reporting utility is designed to accommodate specific customer site requirements. The performance data collected by the data collection facility is prepared by the account reporting utility for use by system maintenance personnel. Reports can be requested for individual user accounts, summaries of user accounts, and system usage. See the OS/32 System Support Utilities Reference Manual.

Through the DISPLAY ACCOUNTING command, the system operator has access to accounting data for one or all tasks in the system.

NOTE

> The OS/32 job accounting subsystem does not report APU usage and time elapsed in a Model 3200MPS System.

1.2.3 Memory Management Subsystem

When a task is loaded, the memory management subsystem dynamically allocates necessary space in memory. OS/32 supports three types of memory:

- Local
- Shared
- System

Local memory is physically contiguous starting from location 0 and contains the operating system, task space, and system space.

Shared memory is located above local memory and is not required to be contiguous. Global task common segments located in shared memory can be used by more than one processor.

System memory is that which is shared by all processors in a Model 3200MPS System. System memory contains both local and shared areas. Local memory is used by the CPU and all APUs.

Local memory is allocated on a first fit basis when sufficient memory is available for a specific task. Free segments are allocated in ascending address order. When no space is available for a task, the memory manager determines which tasks are to be rolled out to ensure that higher priority tasks take precedence. When memory becomes free, adjacent areas are merged together to minimize search time and to provide large free blocks of memory for bigger tasks. System task space is also maintained by the memory manager and is dynamically allocated when a task or device structure is built.

The memory manager maintains task space through free and allocated lists. Segments are allocated dynamically on a first fit basis by searching the free lists. When free task space is allocated to a segment, it is removed from the free list and connected to the allocated list. This list is called the segment control list (SCL). Similarly, whenever a segment is released, its memory space is removed from the allocated list and connected to (or merged into) the free list.

1.2.4 Timer Management Subsystem

The timer management subsystem provides tasks with a set of timer management/maintenance services. These services control all time dependent functions (e.g., time-slicing, I/O, job accounting, and file dating) through the universal clock.

The following timer queues are maintained by the timer management subsystem:

- Time of day
- Device timeout
- Communications device timeout
- Interval timer

There are several timer routines that service these queues. Entries are placed on the time of day queue and the interval timer queue as a result of supervisor call 2 (SVC 2) timer requests. The control blocks on the time of day queue are referred to as timer queue elements (TMQs). The interval timer queue has the same format as the time of day queue but is maintained as a separate queue.

The universal clock consists of a line frequency clock (LFC) and a precision interval clock (PIC). In a 60Hz system, the LFC generates an interrupt every 8.3 milliseconds, or 120 times per second. In a 50Hz system, the LFC generates an interrupt every 10 milliseconds or 100 times per second. The PIC interrupts when a task's requested time interval has expired or at intervals of 4,096 milliseconds, whichever is shorter. If the interval terminates or the time of day is reached, the TMQ is removed from system space and a trap is generated, or the task is removed from timer wait.

In a Model 3200MPS System configuration, the real-time support module (RTSM) provides each processor with a 32-bit real-time counter for timing program execution. These counters are incremented every microsecond by an RTSM 1MHz on-board oscillator. The RRTC instruction allows tasks to read the counters. See the Perkin-Elmer Model 3200MPS System Instruction Set Reference Manual for more information.

1.2.5 File Management Subsystem

The OS/32 file management subsystem stores and retrieves information for a task on secondary storage devices (disks, floppy disks, etc.). The file manager partitions this storage into smaller areas, called files, that can be used by tasks for data and program storage. In addition, the file manager provides tasks with the following support services for file management:

Allocate	creates a file by allocating space on a secondary storage device.
Delete	removes a file from a secondary storage device.
Rename	changes the name of a file.
Open	assigns an lu to a file.
Close	cancels the lu assignment.
Fetch attributes	examines the attributes of a file.
Checkpoint	ensures that all data in an output buffer is written to a secondary storage device.
Software density selection	selects recording density for 6250 bpi magnetic tape drives.

1.2.6 Input/Output (I/O) Subsystem

The I/O subsystem provides a uniform programming interface between the task and external devices. I/O operations can occur in the following task modes:

Wait	halts execution until data transfer is completed.
Proceed I/O	continues task execution during data transfer.
Halt I/O	allows the task to cancel previous proceed I/O requests.
Queued I/O	allows a task to queue I/O requests to a busy device and continue execution until the device is free.

A task trap mechanism can be used to report each completed I/O operation. Wait-only and test I/O functions allow the task to synchronize its execution with the completed I/O operations.

1.2.7 Error Recording Subsystem

The error recording subsystem logs all data on disk errors for the error reporting utility, which analyzes the data and generates reports.

OS/32 memory error recording software supports the memory error log hardware of the Perkin-Elmer Series 3200 machines. Error log hardware keeps a history of the single bit corrected memory errors. The operating system reads the error log hardware and stores the error information into an internal error log buffer. When the error log buffer is full, its contents are stored on an error recording file with the date and time of the last error recorded. When the error recording file is almost full, a warning message is displayed on the system console indicating that a new error recording file should be allocated or that the error reporting utility should be initiated. The error reporting utility provides reports on the previously recorded error information in the error recording file.

The current error status can be displayed to the system console by using the DISPLAY ERRORS command. The internal error log readout period can be changed by the system operator.

1.2.8 Memory Diagnostics Subsystem

The memory diagnostics subsystem eliminates inoperable memory areas from the system without affecting task execution. It allows the operating system to execute when portions of real memory have been removed (holes) or when a part of the system is powered down for maintenance. Memory can be tested, marked on, and marked off through the operator MEMORY command or when the operating system is initialized.

The marked off areas are noted as allocated in the memory map. Memory is marked on when previously marked off memory is to be used again. If an unrecoverable memory error occurs during task execution on a Perkin-Elmer Series 3200 processor, the operating system automatically marks off the area occupied by the task.

1.2.9 Loader and Segmentation Subsystem

The OS/32 resident loader is responsible for loading tasks, reentrant libraries, task common (TCOM) segments, and partial images. These tasks and segments must have been built by Link. Each task image generated by Link contains information related to the task (e.g., task options, size, libraries referenced) in a record called the loader information block (LIB).

The OS/32 resident loader uses this information to generate data areas, set the task options, create segment tables for the tasks, and map the task segments.

All user tasks (u-tasks) in OS/32 are built as though they were loaded at physical address 0 in memory. The relocation/protection hardware automatically relocates the task addresses at run-time by using the task segment table. This process is totally transparent to the user.

The loader is also responsible for creating the task environment; allocating roll files; creating, maintaining, and deleting segment tables; maintaining a segment control list; and mapping and unmapping partial images.

The task image can be divided into pure and impure segments by specifying the SEGMENTED task option when the task is built by Link. Regardless of the number of times a task is loaded, the loader will allow only one copy of the task's pure segment in memory at any one time. A separate copy of the task's impure segment is loaded each time the task is loaded. The relocation/protection hardware ensures the integrity of pure segments by allowing only read and execute only access privileges to those segments.

Access to task common is achieved mnemonically in FORTRAN or assembly programs. The linkages are resolved by Link. Link commands are also used to request read, write, and execute privileges for task common blocks. See the OS/32 Link Reference Manual for more information.

1.2.10 Basic Data Communications Subsystem

The basic data communications subsystem provides a software interface between tasks and common carrier facilities. Basic data communications facilities allow the user to access remote terminals or computers as though they were locally attached peripherals. For example, with OS/32 Data Communications software, a task performs I/O to a remote terminal in the same manner as I/O to a local device.

In addition to providing device independent (logical I/O) access to the task, the subsystem provides a device dependent I/O capability that allows the systems programmer to tailor a communications package to a particular installation. Such a package can include device dependent and device independent support of asynchronous line devices as well as device dependent support of binary synchronous lines.

The OS/32 Basic Data Communications software support package is required for all 32-bit communications products; e.g., HASP, 2780/3780 Remote Job Entry, and the ZDLC Channel Terminal Manager, which supports the SDLC, HDLC, and ADCCP protocols.

1.2.11 Console Monitor Subsystem

The console monitor subsystem processes all I/O requests directed to the system console device and the system log device from all tasks including the command processor task. The console driver is responsible for intercepting system console I/O requests and for directing them to the console monitor or to another monitor task such as MTM. All I/O operations between the system console and tasks running under MTM are routed to the user's terminal through MTM and not through the console monitor.

When a command is issued from the system console, the console monitor issues an SVC 6 to the command processor notifying it of a command to be processed. The command processor interprets the command and issues an SVC 6 to the console monitor indicating that it is ready to accept another command.

The console driver is a part of the OS/32 I/O subsystem and is the module that intercepts I/O requests to the system console, processes them, and gives them to MTM or to the console monitor to do the actual I/O.

The console monitor is the first task dispatched at OS/32 initialization. The console monitor initializes both itself and the dummy device control block (DCB) used by the console driver to intercept requests from the system console. The monitor then issues an SVC 6 to start the command processor.

1.2.12 Command Processor Subsystem

The command processor subsystem accepts commands from the system console monitor, decodes them, and calls the appropriate executor. Commands can be input to the command processor by entering them directly through the system console or issuing them through a foreground task which uses the system console as an interactive I/O device. Commands input from a foreground task are executed by the command processor in the same manner as commands entered from the system console. If an error occurs during execution of a command, the command processor outputs an error message to the console.

An extension to the command processor, the command substitution system (CSS) allows commonly performed sequences of operations to be executed with one command. The CSS routines provide the user with the ability to build, execute, and control files of operator and MTM commands. The user establishes command files that are called from the user console and executed in the user defined sequence. In this way, complex operations can be carried out by the user with few commands. These commands are analogous to macro instructions in assembly language.

The command substitution system (CSS) provides a set of logical CSS commands to conditionally control the precise sequence of commands to be executed. Parameters can be passed as part of a CSS call so that general sequences can be written that take on specific meaning only when the parameters are substituted. Other calls to CSS files can be imbedded within a CSS file (nested calls).

The command processor normally runs at the second highest priority level after the console monitor in OS/32. This task is strictly trap driven and responds to the SVC 6 task queue parameter calls from the console monitor to service a command request. When the command is processed, it signals the console monitor for a new command read via an SVC 6 queue parameter call and then enters into a trap wait state. The command processor priority can be decreased by the operator ATTN command. This command can be used in a real-time application environment to allow a task to run at a higher priority than the command processor.

1.2.13 System Initialization Subsystem

After the operating system is loaded, the system initialization subsystem initializes the memory diagnostics subsystem, error recording subsystems, and system control blocks and tables in memory. It then dispatches the console monitor which readies the command processor to accept commands from the system console.

1.2.14 Internal Interrupt Subsystem

The internal interrupt subsystem is responsible for:

- handling illegal instruction faults,
- handling arithmetic faults,
- detecting memory faults,
- handling system queue service interrupts,
- handling relocation/protection hardware faults,
- handling data format/alignment faults,
- handling power fail and power restore conditions,
- restoring an interrupted task to its previous program state upon resumption of the task,
- handling parameter block errors,

- handling illegal SVCs and SVC interrupts,
- handling machine malfunction interrupts, and
- performing memory image dumps.

Processor dependent interrupt handlers comprise the internal interrupt subsystem. The internal interrupt subsystem does not support external I/O interrupts. External interrupts are handled by the appropriate device drivers.

On a Model 3200MPS System, the CPU handles all fault conditions or interrupts that occur during execution of a task on an APU. Thus, the APU can execute another task while the CPU is handling the fault or interrupt.

1.2.15 Optional User Supervisor Call (SVC) Subsystem

SVC 14 is provided as an optional SVC that can be defined by the user. On execution, the task receives a task trap for SVC 14. See the OS/32 Application Level Programmer Reference Manual for information on how to implement the SVC 14 trap feature.

1.2.16 Floating Point Subsystem

A task has optional access to single and/or double precision floating point instructions under OS/32. Floating point instructions can be executed through hardware or simulated by software. Those systems that do not support floating point options handle all floating point instructions as illegal instructions.

CHAPTER 2
PRIVILEGED TASKS

2.1 INTRODUCTION

In a multi-user system, improper use of certain machine instructions, called privileged instructions, can have a detrimental effect on system integrity. Privileged instructions include storage protection setting, interrupt handling, timer control, input/output (I/O), and some processor status-setting instructions. To prevent accidental or intentional misuse of these instructions, OS/32 provides a privileged operating state in which tasks can execute these instructions. In addition to the privileged operating state, OS/32 also provides a privileged task state in which tasks can access the file account and bare disk OS/32 supervisor routines.

Only privileged tasks can execute in a privileged operating or task state. OS/32 allows three types of privileged tasks:

- executive tasks (e-tasks),
- privileged user tasks (u-tasks), and
- diagnostic tasks (d-tasks).

A task can be linked as a privileged task by specifying one or more of the following task options in the Link OPTION command:

```
    ETASK, ACPRIVILEGE, DISC, DTASK
```

See the OS/32 Link Reference Manual.

This chapter describes the privileges that are available to each type of privileged task through the Link OPTION command.

2.2 EXECUTIVE TASKS (E-TASKS)

E-tasks run with the memory address relocation/protection hardware turned off and are allowed to execute all instructions provided by the hardware. E-tasks always have file account and bare disk privileges. In addition, e-tasks can execute code that modifies or enhances the OS/32 system software. For example, an e-task can modify one of the system modules to enhance an existing OS/32 feature. E-tasks can also function as drivers that support nonstandard peripheral devices. A task can be linked as an e-task by specifying the ETASK task option in the Link OPTION command.

The following sections detail the programming considerations that must be taken into account when writing e-tasks.

2.2.1 Writing Executive Tasks (E-Tasks)

Because e-tasks execute in a privileged state, certain precautions must be taken when e-tasks are programmed.

When an e-task is executing, no memory address protection or relocation is provided and all interrupts are enabled. The task has access to all machine instructions and memory address space in the system. In addition, the e-task can access system tables and control information via the system pointer table (SPT). The address of the SPT is stored in the halfword at location X'62' in memory.

Link builds the image for an e-task as if it were loaded at absolute location zero. The loader, however, is free to load the e-task into any available memory location. Therefore, an e-task must be coded as if it were positionally independent; an e-task must not contain relocatable code.

Because Link relocates e-task addresses to absolute zero, e-tasks cannot assemble code containing address constants as shown in the following example.

Example:

```
SVC7BLK  DB     X'80',7
         DAC    ADDR
```

| An e-task must dynamically set the addresses required by the task.

To reference addresses in the +16kb range, use the following technique:

```
    LA     UE,BUFSTART
    LA     UF,BUFEND
    LA     U3,SVC1PBK
    STM    UE,SVC1.SAD(U3)
    SVC    1,0(U3)
```

References to addresses exceeding the 16kb range can be made in the following manner.

Example:

```
    BASE      LA     U4,BASE
              LA     UE,BUFSTART-BASE(U4)
              LA     UF,BUFEND-BASE(U4)
              LA     U3,SVC1BLK-BASE(U4)
              STM    UE,SVC1.SAD(U3)
              SVC    1,0(U3)
```

E-tasks smaller than 16kb must use the no RX3 (NORX3) (CAL/32) instruction to force all RX instructions to the RX1 or RX2 format. The tasks must not contain any RX1 or RX3 instructions with relocatable addresses. See the Common Assembly Language/32 (CAL/32) Programming Reference Manual.

2.2.2 OS/32 Data Structures Used by Executive Tasks (E-Tasks)

OS/32 provides a data structure macro library that contains macro routines for building OS/32 data structures in the e-task address space. Field names for the data structures are also included. The OS/32 data structure macro library is stored in file SYSSTRUC.MLB. Table 2-1 lists the OS/32 data structures available to e-tasks and their corresponding macros.

Using the OS/32 e-task capability and the data structures available to e-tasks, the system level programmer can incorporate changes or add user written modules to the source of the OS/32 system modules supplied by Perkin-Elmer.

TABLE 2-1 OS/32 DATA STRUCTURES USED BY E-TASKS

MACRO	DATA STRUCTURE
$ACB	Directory access control block
$AOPT	APU options
$APB	Auxiliary processor block
APB	$APB,$APRC,$APS,$AOPT
$APRC	Passback reason codes and equates
$APS	APB status codes and equates
$APST	APU status codes and error codes and equates
$ATF	Accounting transaction file
$AUF	MTM authorized user file
$CCB	Channel control block (CCB)
$CTX	User task context block
$DATB	Device attributes equates
DCB	$PDCB,$DDCB,DCB EQUATE,$DFLAG,$DATB,$DXFL
$DDCB	Device dependent device control block (DCB)
$DDE	Error log data structure
$DFLG	DCB flags
DIR	Primary directory entry
$DXFL	Disk extended flags
$EMIL	System milestone recording entries
$EFMG	Bulk device error recording entries
$EREGS	16 executive registers (E1=register 1)
$ERRC$	$GERC, $EFMG, $ESYS, $EMIL, $MERC
$ESYS	System error recording entries
$EVN	Event node
$FCB	File control block (FCB)
FCB	FCB and FCB flags
$FDE	Free block descriptor entry
$FFLG$	FCB flags
$FD	File descriptor (fd)
$GERC	General error recording information
$HB	Help subroutine argument block
INTCPARM	SVC intercept information
$ICB	Intercept control block
$IOB	I/O block
IOB	I/O and I/O flags
$IOBF	I/O block flags
$IOH	I/O handler list
$IPCB	Intercept path control block
$IRCB	interceptcontrol block
$IVT	Initial value table

TABLE 2-1 OS/32 DATA STRUCTURES USED BY E-TASKS (Continued)

MACRO	DATA STRUCTURE
$LIB	Loader information block (LIB)
LIB	LIB and loader options
$LPMT	Logical processor mapping table
$LOPT	Loader options
$LSG	Load segment
$LTCB	Loader task control block (TCB) redefinitions
$MAGDCB	Magnetic tape DCB
$MERC	Memory error recording entry
$MTMSTE	MTM terminal and task status and modes
$OCB	Overlay control block
$ODT	Overlay descriptor table structure
$ORT	Overlay reference
$PDCB	Primary (device independent) DCB
$PFCB	Private FCB
$PSDCB	Pseudo DCB structure (device dependent)
$PSTCB	Pseudo task control block
$PSW	Program status word (PSW)
$QH	SVC intercept queue handler structure
$RCTX	RS/RSA context block
$REGS$	$SOPT, $UREGS, $EREGS, $RREGS, $PSW
$RLST	Roll selection list
$RREGS	16 general user registers (R1 = register 1)
$RSARCPY	Reentrant system state alternate save area
$S1XO	SVC 1
$SDCB	Pseudo print device control block structure
$SD	Send data message block
$SDE	Segment descriptor element
$SOPT	System options
$SPCMSG	Spooler message structures
$SPT	System pointer table (SPT)
$SPTE	SPT extern definitions
$SPOL	Spooler message
$STE	Segment table entries
$SPR	Segment privilege flags

TABLE 2-1 OS/32 DATA STRUCTURES USED BY E-TASKS (Continued)

MACRO	DATA STRUCTURE
$SVC1	Supervisor call 1 (SVC 1)
$SVC1$	SVC 1 and SVC 1 error codes
$SVC1ERR	SVC 1 error codes
$SVC4	System SVC - reserved
$SVC5	SVC 5 parameter block
$SVC6	SVC 6 parameter block
$SVC7	SVC 7 parameter block
$SVC7SPL	Spooler SVC 7 parameter block
$SVC13	SVC 13 parameter block
$SVC13$	$SVC13,$APST
$SYP	System space structure
$$SPT	SPT table definitions
$TABL$	Structure/extern generating macro
$TCB	Task control block,$SDE,$IOB$,$TCB,$CTX
TCB	$TCB,$TOPT,$TSTT,$TWT,$TLFL,$PSTCB,$OCB,$TQE, $TFL,$TPRC,TQH
$TERMUSR$	MTM terminal user block
$TKQ	Task queue head
$TLF	Task control block flags
$TLFL	Logical unit table of flags
$TMQ	Timer queue entry (TQE)
$TOPT	Task options flags
$TPRC	Task passback codes
$TQE	Task event queue entry
$TQ27	SVC 2, code 27 parameter block
$TQH	Task event queue header
$TSTT	Internal task status flags
$TSW	Task status word (TSW)
$TTB	APU trap block
$TWT	Task wait status flags
$UDL	User dedicated locations (UDL)
UDL	UDL and TSW
$UREGS	16 general user registers (U1 = register 1)
$VD	Volume descriptor
$VFCHARS	Vertical forms characters
$VFDCB	Common VFC DCB structure
$WAP	Read/write access matrix header structure

2.3 PRIVILEGED USER TASKS (U-TASKS)

Like nonprivileged u-tasks, privileged u-tasks run with the memory address relocation/protection hardware enabled and are restricted to a subset of instructions known as nonprivileged instructions. If a u-task attempts to execute a privileged instruction, it causes an illegal instruction fault. However, unlike nonprivileged u-tasks, privileged u-tasks have file account and bare disk privileges. File account privileges allow tasks to specify an account number in the file account/class field of a file descriptor. Bare disk privileges allow tasks to perform I/O operations to a bare disk device. See Chapter 3.

A u-task acquires file account and bare disk privileges by specifying the ACPRIVILEGE and DISC task options, respectively, in the Link OPTION command when the task is built.

2.4 DIAGNOSTIC TASKS (D-TASKS)

D-tasks, like e-tasks, can execute all instructions provided by the hardware. However, like u-tasks, d-tasks run with the memory address relocation/protection hardware enabled and execute in the nonprivileged task state. D-tasks are designed for use in diagnostic applications, loading WCS, and direct execution of I/O instructions.

A task can be linked as a d-task by specifying the DTASK task option in the Link OPTION command. To execute in the privileged task state, a d-task must be built with the ACPRIVILEGE and DISC task options enabled.

CHAPTER 3
OS/32 SUPERVISOR CALLS (SVCs) FOR SYSTEM LEVEL PROGRAMMING

3.1 INTRODUCTION

OS/32 provides a number of system services for designing system level control programs. Access to these services is restricted to privileged tasks and/or by certain hardware requirements. Table 3-1 lists the OS/32 SVCs that access these system level services and the restrictions for their use. The following sections describe each system level SVC in detail.

TABLE 3-1 OS/32 SVCs FOR SYSTEM LEVEL PROGRAMMING

SVC	RESTRICTED TO
SVC 0: User written SVC	Modification of OS/32 required
SVC 2 code 0: Make journal entries	Executive tasks (e-tasks) only
SVC 2 code 14: Internal reader	Foreground tasks loaded from system console
SVC 2 code 26: Fetch device name	E-tasks only
SVC 2 code 27: Memory management	Tasks running on memory address translator (MAT) machines only
SVC 6: System task release	System tasks (.CMDP, .CSL, .MTM, and .SPL)
SVC 7: Extended close	E-tasks or privileged user tasks (u-tasks) and privileged diagnostic tasks (d-tasks) with bare disk (DISC) task option enabled

TABLE 3-1 OS/32 SVCs FOR SYSTEM LEVEL PROGRAMMING (Continued)

SVC	RESTRICTED TO
SVC 7: Fetch attributes for bare disk devices	E-tasks or privileged u-tasks and privileged d-tasks with the bare disk (DISC) task option enabled
SVC 7: Device rename and reprotect	E-tasks only
SVC 7: Bare disk assignment	E-tasks or privileged u-tasks and privileged d-tasks with bare disk (DISC) task option enabled.
SVC 13 code 0 and code 1	No restrictions. Available to all tasks running on the Model 3200MPS System.
SVC 13 code 2 and code 3: Auxiliary processing unit (APU) control	Tasks running on the Model 3200MPS System that have APU control (APCONTROL) and APU mapping (APMAPPING) task options enabled.

3.2 SVC 0: USER WRITTEN SUPERVISOR CALL (SVC)

SVC 0 is reserved for user written OS/32 executor routines. Before writing an executor routine that can be called by SVC 0, the operating system must be modified. This modification can be done dynamically at run-time by an e-task. Note, however, that the SVC executor table contains only halfword entries; therefore, the first instruction of the executor routine called by SVC 0 must lie within the first 64kb of physical memory.

3.3 SVC 2 CODE 0: MAKE JOURNAL ENTRIES

SVC 2 code 0 makes an entry into the system journal from an e-task. The system journal provides a method to trace back important events (SVCs, input/output (I/O) operations, task switching) that occurred during system operation. For example, the journal is useful for tracing the cause of a system failure. The parameter block format for SVC 2 code 0 is shown in Figure 3-1.

```
 ----------------------------------------------------------
|0(0)                         |2(2)                        |
|            Code             |        Journal code        |
|                             |                            |
|----------------------------------------------------------|
|4(4)                                                      |
|                        Value 1                           |
|                                                          |
|----------------------------------------------------------|
|8(8)                                                      |
|                        Value 2                           |
|                                                          |
|----------------------------------------------------------|
|12(C)                                                     |
|                        Value 3                           |
|                                                          |
|----------------------------------------------------------|
|16(10)                                                    |
|                        Value 4                           |
|                                                          |
 ----------------------------------------------------------
```

```
        SVC   2,parblk
         .
         .
         .
parblk  DC    H'0'
        DC    H'journal code'
        DC    F'value 1'
        DC    F'value 2'
        DC    F'value 3'
        DC    F'value 4'
```

Figure 3-1 SVC 2 Code 0 Parameter Block Format and Coding

During execution, a logical OR operation is performed on a mask and the journal code to indicate that the entry originates from an SVC 2 code 0, rather than from within the system. The value 1, 2, 3, and 4 fields of the parameter block are stored following the journal code and calling task name in the journal. These values can contain any desired information to be preserved for system debugging.

NOTE

This call has an effect only if the journal is included in the system at (source) system generation (sysgen).

3.4 SVC 2 CODE 14: INTERNAL READER

SVC 2 code 14 allows a foreground task loaded from the system console to invoke operator and command substitution system (CSS) commands. These commands are sent to the command processor where they are executed as if they were entered from the system console. SVC 2 code 14 provides two options for sending commands to the command processor. Option 0 allows the user to place the commands directly in the task command buffer field of the SVC 2 code 14 parameter block. Option 1 allows the user to store the commands in a task command buffer located on a fullword boundary within the task's address space. The address of this buffer is placed in the parameter block.

SVC 2 code 14 will transfer the commands in a task command buffer until the end of the buffer is reached. The parameter blocks for both SVC 2 code 14 options are described in the following sections.

3.4.1 Parameter Block for Option 0

The parameter block format for option 0 of SVC 2 code 14 is shown in Figure 3-2.

```
 -----------------------------------------------------------
|0(0)          |1(1)          |2(2)                         |
|   Option     |     Code     |           Status            |
|              |              |                             |
|-----------------------------------------------------------|
|4(4)                         |6(6)    Maximum system       |
|        User command         |        command buffer       |
|        buffer length        |            length           |
|-----------------------------------------------------------|
|8(8)                                                       |
|                                                           |
|                                                           |
|-----                                                 -----|
|12(C)                                                      |
|                            Task                           |
|                           command                         |
|-----                      buffer                     -----|
|16(10)                                                     |
|                                                           |
|                                                           |
|----                                                  -----|
|                                                           |
|                                                           |
|                                                           |
|                                                           |
|                                                           |
|                                                           |
 -----------------------------------------------------------
```

```
          SVC    2,parblk
           .
           .
           .
 parblk   DB     0,14,0,0
          DC     H'user command buffer length'
          DC     H'0'
          DC     'operator commands'
```

Figure 3-2 SVC 2 Code 14 Parameter Block Format and Coding for Option 0

This parameter block can be up to 1,032 bytes long, fullword boundary aligned, and located in a task writable segment. A general description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that contains a value of 0 to indicate that the task command buffer is contained in the SVC 2 code 14 parameter block.
Code	is a 1-byte field that contains the decimal value 14 to indicate code 14 of SVC 2.
Status	is a 2-byte field that receives a status code indicating the status of the SVC processing. See
User command buffer length	is a 2-byte field specifying a decimal number indicating the maximum length allowed for the user command buffer.
Maximum system command buffer length	is a 2-byte field to which the operating system returns the system command buffer length established by CMDLEN at sysgen. This value is returned only for status code X'0003'. See Table 3-2.
Task command buffer	is a variable length field with a maximum length of 1,024 bytes. This field contains the commands to be sent to the command processor.

3.4.2 Parameter Block for Option 1

The parameter block format for option 1 of SVC 2 code 14 is shown in Figure 3-3.

```
 -----------------------------------------------------------
|0(0)          |1(1)          |2(2)                         |
|    Option    |     Code     |           Status            |
|              |              |                             |
|-----------------------------------------------------------|
|4(4)                         |6(6)    Maximum system       |
|        User command         |        command buffer       |
|        buffer length        |            length           |
|-----------------------------------------------------------|
|8(8)                                                       |
|                      Buffer address                       |
|                                                           |
 -----------------------------------------------------------
```

```
         SVC   2,parblk
          .
          .
          .
parblk   DB    1,14,0,0
         DC    H'user command buffer length'
         DC    H'0'
         DAC   BUFADR
```

Figure 3-3 SVC 2 Code 14 Parameter Block Format and Coding for Option 1

This parameter block is 12 bytes long, fullword boundary aligned, and located in a task writable segment. A general description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that contains a value of 1 to indicate that the parameter block contains the address of the task command buffer.
Code	is a 1-byte field that contains the decimal value 14 to indicate code 14 of SVC 2.
Status	is a 2-byte field that receives a status code indicating the status of the SVC processing. See Table 3-2 for a list of the SVC 2 code 14 status codes.
User command buffer length	is a 2-byte field specifying a decimal number indicating the maximum length allowed for the task command buffer.

Maximum system command buffer	is a 2-byte field to which the operating system returns the system command buffer length established by CMDLEN at sysgen. This value is returned only for status code X'0003'. See Table 3-2.
Buffer address	is a 4-byte field specifying the address of the task command buffer. This buffer must be located on a fullword boundary within the task's address space.

3.4.3 SVC 2 Code 14 Status Codes

The status codes for each of the SVC 2 code 14 options are listed in Table 3-2.

TABLE 3-2 SVC 2 CODE 14 STATUS CODES

CODE	MEANING
X'0000'	Successful completion - commands sent to command processor for execution.
X'0001'	No free internal reader buffers available.
X'0002'	Option error - invalid option specified for SVC.
X'0003'	User specified length of command buffer incorrectly. The length of the maximum allowed system command buffer is returned to the maximum system command length field.
X'FFFF'	No internal reader command buffers defined.

3.4.4 Programming Considerations

Support for the internal reader must be included in the system at sysgen. This is accomplished through the SYSGEN/32 command, IREADER. See the OS/32 System Generation (SYSGEN/32) Manual. If the internal reader is not included at sysgen, an attempt to execute an SVC 2 code 14 results in an execution error and an illegal SVC message is sent to the user console.

The internal reader requires a set of buffers to receive the commands sent to it by SVC 2 code 14. The OS/32 operator command, IRBUFFER, is used to create command buffers for the internal reader. See the OS/32 Operator Reference Manual. The IRBUFFER command can also be used to increase the number of command buffers when no free buffers are available (status code X'0001'). IRBUFFER can be used at any time if support for the internal reader has been generated into the system.

The following program demonstrates the use of SVC 2 code 14.

Sample SVC 2 code 14 program:

```
SVC214    PROG  SVC 2,14 EXAMPLE
          SVC   2,COMMAND0                SEND COMMAND
          LH    0,COMMAND0+2              WAS IT SUCCESSFUL?
          BNZ   STOP                      NO  - ERROR
          SVC   2,COMMAND1                SEND COMMAND
          LH    0,COMMAND1+2              WAS IT SUCCESSFUL?
          BNZ   STOP                      NO  - ERROR
          SVC   3,0                       EOT

STOP      EQU   *
          SVC   2,PAUSE                   PAUSE
          SVC   3,0                       EOT


          ALIGN 4
PAUSE     DB    0,1,0,0


          ALIGN 4
COMMAND0  DB    0,14,0,0                  DIRECT COMMAND BUFFER
          DC    Z(4)
          DCX   0
          DC    C' D M '


          ALIGN 4
COMMAND1  DB    1,14,0,0                  INDIRECT COMMAND BUFFER ADDRESS
          DC    Z(CMDBUFFE-CMDBUFF)
          DCX   0
          DC    A(CMDBUFF)
          ALIGN 4
CMDBUFF   DC    C'$WR ** CSS CALL BY IREADER ***;    CALLCSS.CSS    '
CMDBUFFE  EQU   *
          END
```

3.5 SVC 2 CODE 26: FETCH DEVICE NAME

SVC 2 code 26 searches for a user supplied volume name in the volume mnemonic table and returns the name of the device on which that volume is mounted. The format for the SVC 2 code 26 parameter block is shown in Figure 3-4.

```
 -----------------------------------------------------------
|0(1)          |1(1)          |2(2)          |3(3)          |
|   Reserved   |     Code     |     User     |     User     |
|              |              |  register 1  |  register 2  |
 -----------------------------------------------------------

           SVC    2,parblk
            .
            .
            .
  parblk   ALIGN 4
           DB     0,26
           DB     user register number 1
           DB     user register number 2
```

Figure 3-4 SVC 2 Code 26 Parameter Block Format and Coding

This parameter block is four bytes long, fullword boundary aligned, and does not have to be in a writable segment of the task. The fields are described as follows.

Fields:

Reserved	is a 1-byte field that must contain a value of 0 to indicate no options for this call.
Code	is a 1-byte field that must contain the decimal value 26 to indicate code 26 of SVC 2.
User register 1	is a 1-byte field that must contain a user-specified register number. The specified register contains a pointer to a fullword containing a 4-character volume name.

User register 2	is a 1-byte field that must contain a user specified register number. The specified register contains the address of the area receiving the device name. This area is 4 bytes long, fullword boundary aligned, and must be located in a task writable segment.

NOTE

User register 1 and user register 2 can specify the same register number.

Possible condition codes occurring after SVC 2 code 26 execution follow:

Condition Code:

C	V	G	L	
0	0	0	0	Normal termination.
0	1	0	0	Specified volume offline; no fetch occurred.

Example:

```
                .
                .
                LA     R1,MTMVOLN
                LA     R2,MTMDEVN
                SVC    2,FTCHDEVN
                .
                .
                .
                ALIGN 4
       FTCHDEVN DB     0,26
                DB     R1
                DB     R2
```

3.6 SVC 2 CODE 27: MEMORY MANAGEMENT

SVC 2 code 27 allows a task to access and modify entries (except shared ones) within the private segment table (PST) in its task control block (TCB). This SVC can only be called by tasks running on MAT machines. It is used by the virtual task manager (VTM) support routines. The format for the SVC 2 code 27 parameter block is shown in Figure 3-5.

```
 ------------------------------------------------------------
|0(0)           |1(1)           |2(2) User      |3(3) User      |
|    Option     |     Code      |  register 1   |  register 2   |
|  (SV227.OP)   |  (SV227.CD)   |  (SV227.R1)   |  (SV227.R2)   |
|------------------------------------------------------------|
|4(4)                                                        |
|                    A (destination buffer)                  |
|                          (SV227.BF)                        |
 ------------------------------------------------------------

             SVC    2,parblk
              .
              .
              .
             ALIGN 4
    parblk   DB     option,27,reg1,reg2
             DAC    BUFFADR
```

Figure 3-5 SVC 2 Code 27 Parameter Block Format and Coding

This parameter block is 8 bytes long, fullword boundary aligned, and located in a task writable segment. A general description of the parameter block follows.

Fields:

Option (SV227.OP)	is a 1-byte field that contains a decimal number specifying one of the following option codes:

OPTION CODE	FUNCTION EQUATE	MEANING
0	SV227.0	The first byte of each PST entry, starting at PSTE 0 and ending with the last private segment table entry (PSTE) of the task's impure segment (PSTE indicated by TCB.CTOP), is moved sequentially to a byte buffer specified by the SV227.BF field. After a byte is moved, the reference bit (bit 0) of each PSTE is reset.
1	SV227.1	Bits 15-31 of PSTE 0 are added to the user register specified by the SV227.R1 field. The result of the addition is stored in the PSTE identified by the number contained in the register specified by the SV227.R2 field.
2	SV227.2	The value in user register 1 is stored in TCB.UTOP and the value in user register 2 is stored in TCB.CTOP.

Code (SV227.CD)	is a 1-byte field that contains the decimal number 27 indicating code 27 of SVC 2.
User register 1 (SV227.R1)	is a 1-byte field that contains a user specified register number. If option 0 is specified, this field is unused but must be reserved.
User register 2 (SV227.R2)	is a 1-byte field that contains a user specified register number. If option 0 is specified, this field is unused but must be reserved.
A (destination buffer) (SV227.BF)	is a 4-byte field that contains the address of the first byte of a user specified buffer to which the entries in the PST will be copied. This buffer should be located in a task writable segment. This field is omitted for options 1 and 2.

SVC 2 code 27 sets the condition code field in the PSW as follows.

Condition Code:

C	V	G	L	
0	0	0	0	SVC 2 code 27 completed. No errors.
0	0	0	1	Size of PST entry exceeds task allocation of memory. Entry not stored in PST.
0	0	1	0	Illegal PST entry number.
0	1	0	0	Shared bit set in PST entry. This entry cannot be modified.
1	0	0	0	Value of UTOP is greater than value of CTOP.

3.7 SVC 6: SYSTEM TASK RELEASE

The SVC 6 release function can be used by a system task (.CMDP, .CSL, .MTM, and .SPL) to remove another task from a suspended state. After the task is released, it continues execution at the location specified in the SVC6.SAD field of the SVC 6 parameter block. If this SVC is used by other than a system task, the continuation address (SVC6.SAD) is ignored. Figure 3-6 shows the parameter block format and coding for the SVC 6 release function for system tasks.

```
 ------------------------------------------------------------
|0(0)                                                        |
|                                                            |
|                       Name of task                         |
|-----                 receiving SVC6                   -----|
|4(4)                    (SVC6.ID)                           |
|                                                            |
|                                                            |
|------------------------------------------------------------|
|8(8)                                                        |
|                      Function code                         |
|                        (SVC6.FUN)                          |
|------------------------------------------------------------|
|12(C)                        |14(E)                         |
|         Reserved            |          Error status        |
|        (SVC6.TST)           |           (SVC6.STA)         |
|------------------------------------------------------------|
|16(10)                                                      |
|                          Reserved                          |
|                         (SVC6.LU)                          |
|------------------------------------------------------------|
|20(14)                                                      |
|            Continue address of directed task               |
|                        (SVC6.SAD)                          |
|------------------------------------------------------------|
|24(18)                                                      |
                          Reserved
                         (SVC6.TIM)
|44(2C)                                                      |
 ------------------------------------------------------------
```

```
              SVC    6,parblk
                .
                .
                .
              ALIGN 4
     parblk   DC     C'8-byte name of task receiving SVC 6'
              DC     Y'80000080'
              DC     H'0'
              DC     H'0'
              DC     0
              DC     A(START)
              DC     0,0,0,0,0,0
```

Figure 3-6 SVC 6 Release Function Parameter Block Format and Coding

This parameter block must be 48 bytes long, fullword boundary aligned, and located in a task writable segment. A general description of each field in the parameter block follows.

Fields:

Name of task receiving SVC 6 (SVC6.ID)	is an 8-byte field containing the name of the task to which SVC 6 is directed. This name must be left-justified with blanks.
Function code (SVC6.FUN)	is a 4-byte field containing the hexadecimal number Y'80000080' indicating that the release function is to be performed.
Reserved (SVC6.TST)	is a reserved halfword field that must contain a zero.
Error status (SVC6.STA)	is a 2-byte field that receives an error code when an error occurs during SVC 6 execution. If no error occurs, a value of 0 is stored in this field. See the OS/32 Supervisor Call (SVC) Reference Manual for a listing of SVC 6 error codes.
Reserved (SVC6.LU)	is a reserved fullword field that must contain a zero.
Continue address of directed task (SVC6.FUN)	is a fullword field that contains the address at which the directed task is to resume execution after it$is released. If this field contains a zero, execution continues at the location at which the task was suspended. If this option is specified and the task issuing the SVC is not a system task, this field is ignored.
Reserved (SVC6.TIM)	is a reserved 24-byte field that must contain zeros.

3.8 SVC 7: EXTENDED FUNCTIONS FOR PRIVILEGED TASKS

The OS/32 file manager provides additional resource management services to privileged tasks. These services are accessed through the extended function codes of SVC 7. These functions include:

- bare disk assignment,
- fetch attributes for bare disk devices,
- device rename and reprotect, and
- extended close.

The following sections describe how to access the SVC 7 extended functions for privileged tasks.

3.8.1 SVC 7: Bare Disk Assignment

An e-task, privileged u-task, or d-task with bare disk privileges can bypass the file manager and directly assign to a disk device. When such a task issues an SVC 1 I/O request directly to a disk device, the operation is referred to as bare disk I/O and should always be random access. The supported I/O functions are read, write, and test and set.

Direct assignment to a disk device can be performed only by a task that has the bare disk task option enabled. E-tasks always have this option enabled. A u- or d-task is given bare disk privileges by specifying the disk privilege option in the OPTION command (OPTION DISC) when the task is built by Link. However, if the task loader has the e-task option prevented, the privileged task is loaded into memory with the bare disk privilege option changed to the default, no bare disk privilege. Therefore, bare disk I/O cannot be performed by the task.

The SVC 7 parameter block and coding for bare disk assignments are shown in Figure 3-7.

```
 -----------------------------------------------------------
|0(0)                          |2(2) Error    |3(3)          |
|        Function code         | status code  |      lu      |
|         (SVC7.OPT)           | (SVC7.STA)   |  (SVC7.LU)   |
|-----------------------------------------------------------|
|4(4)                                                        |
|                          Reserved                          |
|                         (SVC7.WKY)                         |
|-----------------------------------------------------------|
|8(8)                                                        |
|                      Device mnemonic                       |
|                         (SVC7.VOL)                         |
|-----------------------------------------------------------|
|12(C)                                                       |
|                                                            |
|                                                            |
|------                                                ------|
|16(10)                    Reserved                          |
|                         (SVC7.FNM)                         |
|                                                            |
|------                                                ------|
|24(18)                                                      |
|                                                            |
|                                                            |
 -----------------------------------------------------------
```

```
         SVC    7,parblk
           .
           .
           .
         ALIGN 4
parblk   DC     X'function code'
         DB     0
         DB     lu
         DC     0
         DC     C'4-character device mnemonic'
         DC     0,0,0,0
```

Figure 3-7 SVC 7 Bare Disk Assignment Parameter Block Format and Coding

This parameter block must be 28 bytes long, fullword boundary aligned, and located in a task writable segment. A description of each field in the parameter block follows.

Fields:

Function code (SVC7.OPT)	is a 2-byte field that contains the hexadecimal number indicating the assign function (bit 1 must be set). In addition, the appropriate access privileges (bits 8 through 10) must be set as follows:

- 000 = Shared Read-Only (SRO)
- 001 = Exclusive Read-Only (ERO)
- 010 = Shared Write-Only (SWO)
- 011 = Exclusive Write-Only (EWO)
- 100 = Shared Read/Write (SRW)
- 101 = Shared Read, Exclusive Write (SREW)
- 110 = Exclusive Read, Shared Write (ERSW)
- 111 = Exclusive Read/Write (ERW)

CAUTION

IF THE BARE DISK IS MARKED ONLINE, ONLY ASSIGNMENTS FOR SRO ARE ALLOWED. ANY OTHER ACCESS PRIVILEGE IS REJECTED, AND A PRIVILEGE ERROR STATUS (07) IS GIVEN.

Error status codes (SVC7.STA)	is a 1-byte field that receives an error code when an error occurs during SVC 7 execution. If no error occurs, a value of 0 is return to this field. See the OS/32 Supervisor Call (SVC) Reference Manual for a list of SVC 7 error status codes.
lu (SVC7.LU)	is a 1-byte field that contains a hexadecimal number indicating the logical unit to be assigned to the bare disk device.
Reserved (SVC7.WKY)	is a 4-byte reserved field that must contain a zero.
Device mnemonic (SVC7.VOL)	is a 4-byte field containing the device mnemonic of the bare disk device.
Reserved (SVC7.FNM)	is a 16-byte field that must be reserved with zeros.

3.8.2 SVC 7 Code 0: Fetch Attributes for Bare Disk Devices

The fetch attributes function fetches the attributes of a bare disk device through its assigned logical unit (lu). The write attribute is reset in the attributes halfword field (SVC7.ATRB) of the parameter block if the disk is marked on protected.

This SVC 7 function is available only to tasks with bare disk privileges or to e-tasks. Before issuing this SVC, the task must have the bare disk already assigned to the lu. Figure 3-8 shows the parameter block format and coding for SVC 7 code 0.

```
 -----------------------------------------------------------
|0(0)          |1(1)          |2(2)          |3(3)          |
| Option code  | Device code  |    Status    |      lu      |
|              |              |              |              |
|-----------------------------------------------------------|
|4(4)                         |6(6)                         |
|        Attributes           |        Device number        |
|                             |                             |
|-----------------------------------------------------------|
|8(8)                                                       |
|                          Volume                           |
|                                                           |
|-----------------------------------------------------------|
|12(C)                                                      |
|                           Flags                           |
|                                                           |
|-----------------------------------------------------------|
|16(10)                                                     |
|                           Size                            |
|                                                           |
|-----------------------------------------------------------|
|20(14)                       |22(16)                       |
|     Tracks per cylinder     |     Sectors per track       |
|                             |                             |
|-----------------------------------------------------------|
|24(18)                       |26(1A)                       |
|     Controller address      |       SELCH address         |
|                             |                             |
 -----------------------------------------------------------
```

```
         SVC    7,parablk
          .
          .
          .
         ALIGN  4
parblk   DB     0
         DS     2
         DB     lu
         DS     24
```

Figure 3-8 SVC 7 Code 0 Parameter Block Format and Coding

This parameter block must be 28 bytes long, fullword boundary aligned, and located in a task writable segment. A general description of each field in the parameter block follows.

Fields:

Option	is a 1-byte field that must contain X'00' to indicate the fetch attributes function code.
Device code	is a 1-byte field that receives the device code of the bare disk device.
Status	is a 1-byte field that receives the return status of the bare disk device.
lu	is a 1-byte field that must contain the logical unit for which attributes are returned.
Attributes	is a 2-byte field that receives the attributes of the bare disk device.
Device number	is a 2-byte field that receives the device address of the bare disk.
Volume	is a 4-byte field that receives the device mnemonic for the bare disk.
Flags	is a 4-byte field that receives the device flags for the bare disk.
Size	is a 4-byte field that receives the number of sectors on the bare disk.
Tracks per cylinder	is a 2-byte field that receives the number of tracks per cylinder on the bare disk.
Sectors per track	is a 2-byte field that receives the number of sectors per track on the bare disk.
Controller address	is a 2-byte field that receives the controller address for the bare disk device.
SELCH address	is a 2-byte field that receives the selector channel address for the bare disk device.

3.8.3 SVC 7: Device Rename

E-tasks can use the SVC 7 rename function to rename devices. The e-task must have the device already assigned to the lu with ERW access privileges.

The SVC 7 parameter block format and coding for renaming devices is shown in Figure 3-9.

```
 ------------------------------------------------------------
|0(0)                           |2(2)  Error   |3(3)          |
|        Function code          |  status code |      lu      |
|         (SVC7.OPT)            |  (SVC7.STA)  |  (SVC7.LU)   |
|------------------------------------------------------------|
|4(4)                                                        |
|                         Reserved                           |
|                        (SVC7.WKY)                          |
|------------------------------------------------------------|
|8(8)                                                        |
|                     Device mnemonic                        |
|                        (SVC7.VOL)                          |
|------------------------------------------------------------|
|12(C)                                                       |
|                                                            |
|                                                            |
|------                                                ------|
|16(10)                   Reserved                           |
                          (SVC7.FNM)

|------                                                ------|
|24(18)                                                      |
|                                                            |
|                                                            |
 ------------------------------------------------------------
```

```
              SVC    7,parblk
               .
               .
               .
              ALIGN 4
     parblk   DC     X'1000'
              DB     0
              DB     lu
              DC     0
              DC     C'4-character device mnemonic'
              DB     0,0,0,0
```

Figure 3-9 SVC 7 Device Rename Parameter Block Format and Coding

This parameter block must be 28 bytes long, fullword boundary aligned, and located in a task writable segment. A description of each field in the parameter block follows.

Fields:

Function code (SVC7.OPT)	is a 2-byte field that contains the hexadecimal number X'1000' indicating that the rename function is to be performed.
Error status codes (SVC7.STA)	is a 1-byte field that receives an error code when an error occurs during SVC 7 execution. If no error occurs, a value of 0 is returned to this field. See the OS/32 Supervisor Call (SVC) Reference Manual for a list of SVC 7 error status codes.
lu (SVC7.LU)	is a 1-byte field that contains a hexadecimal number indicating the logical unit assigned to the device that is to be renamed.
Reserved (SVC7.WKY)	is a 4-byte reserved field that must contain a zero.
Device mnemonic (SVC7.VOL)	is a 4-byte field containing the device mnemonic that is to replace the current device mnemonic in the device mnemonic table.
Reserved (SVC7.FNM)	is a 16-byte field that must be reserved with zeros.

3.8.4 SVC 7: Device Reprotect

E-tasks can use the SVC 7 reprotect function to define the type of access allowed to a device (e.g., read only, write only, etc.). The e-task must have the device already assigned to the lu with ERW access privileges.

The SVC 7 parameter block format and coding for reprotecting devices is shown in Figure 3-10.

```
 ------------------------------------------------------------
|0(0)                           |2(2)  Error   |3(3)          |
|         Function code         | status code  |      lu      |
|          (SVC7.OPT)           |  (SVC7.STA)  |  (SVC7.LU)   |
|------------------------------------------------------------|
|4(4)          |5(5)           |6(6)                         |
|  Write  key  |   Read key    |          Reserved           |
|  (SVC7.WKY)  |  (SVC7.RKY)   |         (SVC7.IRC)          |
|------------------------------------------------------------|
|8(8)                                                        |
|                                                            |
|                                                            |
|------                                                ------|
|                                                            |
~                                                            ~
                          Reserved
~                        (SVC7.VOL)                          ~
|------                                                ------|
|24(18)                                                      |
|                                                            |
|                                                            |
 ------------------------------------------------------------
```

```
          SVC    7,parblk
           .
           .
           .
          ALIGN  4
parblk    DC     X'0800'
          DB     0
          DB     lu
          DB     'write key'
          DB     'read key'
          DC     H'0'
          DC     0
          DB     0,0,0,0
```

Figure 3-10 SVC 7 Device Reprotect Parameter Block Format and Coding

This parameter block must be 28 bytes long, fullword boundary aligned, and located in a task writable segment. A description of each field in the parameter block follows.

Fields:

Function code (SVC7.OPT)	is a 2-byte field containing the hexadecimal number X'0800' indicating that the reprotect function is to be performed.
Error status codes (SVC7.STA)	is a 1-byte field that receives an error code when an error occurs during SVC 7 execution. If no error occurs, a value of 0 is returned to this field. See the OS/32 Supervisor Call (SVC) Reference Manual for a list of SVC 7 error status codes.
lu (SVC7.LU)	is a 1-byte field that contains a hexadecimal number indicating the logical unit assigned to the device that is to be reprotected.
Write key (SVC7.WKY)	is a 1-byte field that contains a hexadecimal number indicating the new write protection keys for the device.
Read key (SVC7.RKY)	is a 1-byte field that contains a hexadecimal number indicating the new read protection keys for the device.
Reserved (SVC7.LRC)	is a 2-byte reserved field that must contain a zero.
Reserved (SVC7.VOL)	is a 20-byte unused field that should be initialized with zeros.

3.8.5 SVC 7 Code X'FF80': Extended Close Function

The extended close function closes an lu and replaces the date and time of allocation and the last write (or write filemark) operation in the disk directory with information stored in the SVC 7 parameter block. This SVC 7 function is available only to e-tasks or privileged u-tasks and d-tasks with the bare disk task option enabled.

Figure 3-11 shows the parameter block format and coding for SVC 7 code X'FF80'.

```
 ------------------------------------------------------------
|0(0)                         |2(2)          |3(3)           |
|       Function code         | Error status |      lu       |
|                             |              |               |
|------------------------------------------------------------|
|4(4)                                                        |
|                  Allocation date/time                      |
|                  (moved into DIR.DATE)                     |
|------------------------------------------------------------|
|8(8)                                                        |
|             Last write operation date/time                 |
|                  (moved into DIR.LUSE)                     |
 ------------------------------------------------------------
```

```
            SVC    7,parblk
             .
             .
             .
            ALIGN 4
   parblk   DC     X'FF80'
            DB     1
            DB     lu
            DC     Y'allocation date/time'
            DC     Y'last write operation date/time'
```

Figure 3-11 SVC 7 Code X'FF80' Parameter Block Format and Coding

This parameter block must be 12 bytes long, fullword boundary aligned, and located in a task writable segment. A general description of each field in the parameter block follows.

Fields:

Function code	is a 2-byte field that contains the function code X'FF80' indicating that the SVC 7 extended close function is to be performed.
Error status	is a 1-byte field that receives an error code when an error occurs during SVC 7 execution. If no error occurs, a value of 0 is returned to this field. If a non-privileged task attempts to execute this SVC, a value of 1 is returned. See the OS/32 Supervisor Call (SVC) Reference Manual for a list of SVC 7 error codes.

Allocation date/time	is a 4-byte field that contains the date and time that is to replace the date and time of allocation in the DIR.DATE field of the disk directory. The format of the date and time must be the same format generated by the DATE.DIR routine in the file manager utility (FMUT) module.
Last write operation date/time	is a 4-byte field that contains the date and time that is to replace the date and time of the last write operation in the DIR.LUSE field of the disk directory. The format of the date and time must be the same format generated by the DATE.DIR routine in the FMUT module.

3.9 SVC 13: AUXILIARY PROCESSING UNIT (APU) CONTROL

SVC 13 provides a task with an interface to the APU in a Model 3200MPS System environment. SVC 13 gives a task the ability to:

- access status information on all APUs in the system,
- direct the flow of tasks to an APU, and
- direct the execution of tasks on an APU.

Table 3-3 lists the SVC 13 function codes which provide these capabilities.

Functions 0 and 1 are available to any task in a Model 3200MPS System. Function 2 is available only to tasks in a Model 3200MPS System that have been linked with the APMAPPING task option. Function 3 is available only to tasks in a Model 3200MPS System that have been linked with the APCONTROL task option. See the OS/32 Link Reference Manual for more information on building a task with these task options set.

TABLE 3-3 SVC 13 FUNCTION CODES

FUNCTION CODE	MEANING
SVC 13 code 0	Fetch logical processor mapping table (LPMT).
SVC 13 code 1	Fetch APU status.
SVC 13 code 2	Execute APU mapping option.
SVC 13 code 3	Execute APU control option.

The following sections describe how to access each of the SVC 13 functions from an application task running on a Model 3200MPS System.

3.9.1 SVC 13 Code 0: Fetch Logical Processor Mapping Table (LPMT)

The LPMT is an OS/32 data structure that defines the relationship between the logical processing units (LPUs) assigned to tasks and the APUs of the system. The LPMT contains one entry for each LPU number. In addition to the LPU entries, the LPMT contains the maximum LPU and APU numbers allowed on the system. Each LPU entry is 1-byte, sequentially arranged, starting with the entry for LPU0 (which is always mapped to the CPU) and ending with the entry for LPUn. The maximum value for n is determined at OS/32 sysgen. See the OS/32 System Generation (SYSGEN/32) Reference Manual.

When an APU is mapped to an LPU, its number is placed into the byte corresponding to the LPU entry in the LPMT. SVC 13 code 0 allows a task to copy the LPMT into a data buffer located in a writable segment of the task's address space. This buffer must begin on a fullword boundary. Figure 3-12 shows the parameter block and coding for SVC 13 code 0.

0(0) Reserved (SV13.OPT)	1(1) Function code (SV13.FUN)	2(2) Reserved (SV13.DOP)	3(3) Reserved (SV13.APU)
4(4) Reserved (SV13.APS)		6(6) Error status code (SV13.ERR)	
8(8) Data buffer start address (SV13.BUF)			
12(C) Buffer used (SV13.USE)		14(E) Length of buffer (SV13.LEN)	

```
          SVC    13,parblk
            .
            .
            .
          ALIGN 4
parblk    DB     0,0,0,0
          DB     H'0'
          DS      2
          DC     H'number of bytes'
```

Figure 3-12 SVC 13 Code 0 Parameter Block Format and Coding

This parameter block must be 16 bytes long, fullword boundary aligned, and located in a task writable segment. A general description of each field in the parameter block follows:

Fields:

Reserved (SV13.OPT)	is a 1-byte unused field that should be initialized to zero.
Function code (SV13.FUN)	is a 1-byte field that must contain the decimal number 0 to indicate code 0 of SVC 13.
Reserved (SV13.DOP)	is a 1-byte unused field that should be initialized to zero.
Reserved (SV13.APU)	is a 1-byte unused field that should be initialized to zero.
Reserved (SV13.APS)	is a 2-byte unused field that should be initialized to zero.
Error status code (SV13.BUF)	is a 2-byte field that receives the execution status of SVC 13 code 0. See Table 3-10 for a list of the SVC 13 status codes.
Data buffer start address (SV13.BUF)	is a 4-byte field that contains the address of a user specified buffer to which the operating system returns an exact copy of the LPU entries in the LPMT, including reserved fields, the sign of each LPU entry, the maximum LPU number, and the maximum APU number. The buffer can be variable in length but must begin on a fullword boundary in a task writable segment.
Buffer used (SV13.USE)	is a 2-byte field that receives the actual number of bytes used in the buffer specified by the SV13.BUF field.
Length of buffer (SV13.LEN)	is a 2-byte field that contains a decimal number indicating the maximum length (in bytes) of the data buffer specified in the SV13.BUF field.

When SVC 13 code 0 is executed, APU assignment information is returned to the user's buffer in the format shown in Figure 3-13.

```
 ----------------------------------------------------------
|0(0)           |1(1)          |2(2)          |3(3)         |
|   Reserved    | Maximum APU  | Maximum LPU  |  LPU entry  |
|               |    number    |    number    |    width    |
|-----------------------------------------------------------|
|4(4)                                                       |
|              LPU to APU mapping array                     |
|                                                           |
 ----------------------------------------------------------
```

Figure 3-13 Data Buffer Format for SVC 13 Code 0

A general description of the fields in the APU mapping data buffer follows:

Fields:

Reserved	is a 1-byte field that is reserved for future use.
Maximum APU number	is a 1-byte field containing the maximum APU number allowed in the system. This number is determined at OS/32 sysgen.
Maximum LPU number	is a 1-byte field containing the maximum LPU number allowed in the system. This number is determined at OS/32 sysgen.
LPU entry width	is a 1-byte field containing the width (in bytes) of each LPU entry in the LPMT.
LPU to APU mapping array	is a variable length array (0:MAXLPU) containing the LPU to APU mapping assignments. The index to the array corresponds to the LPU number. The entry at that index into the array contains the APU ID of the APU mapped to the LPU.

3.9.2 SVC 13 Code 1: Fetch Auxiliary Processing Unit (APU) Status

SVC 13 code 1 allows a task to access information on the status of each APU in a Model 3200MPS System. When executed, SVC 13 code 1 returns the following status information on the specified APU to the requesting task's buffer:

- The names of all tasks in the APU ready queue (or name of task having exclusive rights to the APU)

- The APU processing status
- The status of writable control store (WCS)
- The task currently active on the APU (or the waiting task if the APU is stopped)
- The name of the task with control rights over the APU
- The name of the task with mapping rights over the APU
- The number of LPUs mapped to the APU
- The configuration options set for the APU

Figure 3-14 shows the parameter block format and coding for SVC 13 code 1.

```
 ------------------------------------------------------------
|0(0)          |1(1)          |2(2)          |3(3)           |
|   Reserved   | Function code|   Reserved   |  APU number   |
|  (SV13.OPT)  |  (SV13.FUN)  |  (SV13.DOP)  |  (SV13.APN)   |
|------------------------------------------------------------|
|4(4)                         |6(6)                          |
|     APU hardware status     |      Error status code       |
|         (SV13.APS)          |          (SV13.ERR)          |
|------------------------------------------------------------|
|8(8)                                                        |
|                 Data buffer start address                  |
|                        (SV13.BUF)                          |
|------------------------------------------------------------|
|12(C)                        |14(E)                         |
|         Buffer used         |      Length of buffer        |
|         (SV13.USE)          |          (SV13.LEN)          |
 ------------------------------------------------------------
```

```
         SVC    13,parblk
          .
          .
          .
 parblk  ALIGN  4
         DB     0,1,0
         DC     'APU number'
         DS     4
         DC     A(buffer)
         DS     2
         DC     H'number of bytes in buffer'
```

Figure 3-14 SVC 13 Code 1 Parameter Block Format and Coding

This parameter block must be 16 bytes long, fullword boundary aligned, and located in a task writable segment. A general description of each field in the parameter block follows.

Fields:

Reserved (SV13.OPT)	is a 1-byte unused field that should be initialized to zero.
Function code (SV13.FUN)	is a 1-byte field that must contain the decimal number 1 to indicate code 1 of SVC 13.
Reserved (SV13.DOP)	is a 1-byte unused field that should be initialized to zero.
APU number (SV13.APN)	is a 1-byte field specifying a decimal number (0 through 9) indicating the number of the APU to which this call is directed.
APU hardware status (SV13.APS)	is a 2-byte field that receives the APU response status from the APU processor hardware. See Section 3.9.5 for more information on the status codes returned to this field.
Error status code (SV13.ERR)	is a 2-byte field that receives the execution status of SVC 13 code 1. See Table 3-10 for a list of the SVC 13 status codes.
Data buffer start address (SV13.BUF)	is a 4-byte field containing the address of a data buffer to which SVC 13 is to return the APU status information. The buffer can be variable in length, but it must begin on a fullword boundary and be located in a task writable segment. If the buffer is less than 8 bytes in length, an error code (insufficient buffer space) will be returned and no data will be written to the buffer.
Buffer used (SV13.USE)	is a 2-byte field that receives a decimal number indicating the actual number of bytes used in the buffer specified by the SV13.BUF field.
Length of buffer (SV13.LEN)	is a 2-byte field that contains a decimal number indicating the maximum length (in bytes) of the data buffer specified in the SV13.BUF field.

When SVC 13 code 1 is executed, information on the status of the specified APU is returned to the user's buffer in the format shown in Figure 3-15.

```
 -----------------------------------------------------------
|0(0)               |1(1)               |2(2)               |
|     APU number    |    Number of      |  Number of tasks  |
|                   |   LPUs mapped     |                   |
|-----------------------------------------------------------|
|4(4)                          |                            |
|     APU software status      |        APU options         |
|                              |                            |
|-----------------------------------------------------------|
|8(8)                                                       |
|                                                           |
|                    Active task name                       |
|------                     or                        ------|
|12(C)              Waiting task name                       |
|                                                           |
|                                                           |
|-----------------------------------------------------------|
|16(10)                                                     |
|                                                           |
|                                                           |
|------              Control task name                ------|
|20(14)                                                     |
|                                                           |
|                                                           |
|-----------------------------------------------------------|
|24(18)                                                     |
|                                                           |
|                                                           |
|------               Mapping task name               ------|
|28(1C)                                                     |
|                                                           |
|                                                           |
|-----------------------------------------------------------|
|32(20)                                                     |
|                                                           |
|                   Ready task name (1)                     |
|------                     or                        ------|
|36(24)            Exclusive task name                      |
|                                                           |
|                                                           |
|-----------------------------------------------------------|
~                            .                              ~
                             .
~                            .                              ~
|-----------------------------------------------------------|
|                                                           |
|                                                           |
|                                                           |
|------              Ready task name (N)              ------|
|                                                           |
|                                                           |
|                                                           |
 -----------------------------------------------------------
```

Figure 3-15 Data Buffer Format for SVC 13 Code 1

A general description of the fields in the data buffer for SVC 13 code 1 follows.

Fields:

APU number	is a 1-byte field containing the number of the APU to which the status information applies.
Number of LPUs mapped	is a 1-byte field containing the number of LPMT entries mapped to the specified APU.
Number of tasks	is a 2-byte field containing the total number of tasks waiting in the specified APU's ready queue. Any task currently executing on the APU is not included in the total.
APU software status	is a 2-byte field that contains the current software status of the APU, the status after the last power fail, and the state of the WCS for the specifed APU. Figure 3-16 shows the APU software status field. See Table 3-4 for the bit definitions for this field.

```
 ---------------------------------------------------------------------------
| WCS   |          | Last power  |                            | Current   |
|Status | Reserved |failure status|        Reserved            | status    |
 ---------------------------------------------------------------------------
 Bits:
 0     1 2        4 5             7 8                     12 13          15
```

Figure 3-16 Format of APU Software Status Field Returned to Task Buffer

TABLE 3-4 BIT DEFINITIONS FOR APU SOFTWARE STATUS FIELD RETURNED TO TASK BUFFER

APU STATUS	BIT POSITION	BIT SETTING AND MEANING
WCS state	0	1 = WCS initialized 0 = WCS not initialized
	1	1 = WCS loaded 0 = WCS not loaded
Reserved for future use	2-4	
Status after last power failure	5 6 7	111 = APU on and waiting for task 110 = APU on exclusive and waiting for task 011 = APU on 010 = APU on exclusive 001 = APU off
Reserved for future use	8-12	
Current status	13 14 15	111 = APU on and waiting for task 110 = APU on exclusive and waiting for task 011 = APU on 010 = APU on exclusive 001 = APU off 000 = APU disabled

APU options	is a 2-byte field that is set according to the configuration options in effect for the specified APU. See Table 3-5 for the bit definitions for this field.

TABLE 3-5 APU OPTIONS FIELD BIT DEFINITIONS

APU CONFIGURATION OPTION	BIT POSITION	BIT SETTING AND MEANING
WCS	0	0 = APU has WCS 1 = APU has no WCS
Floating point support	1	0 = APU has floating point support 1 = APU has no floating point support
Trap block wait	2	0 = APU will continue processing during CPU fault handling 1 = APU will stop processing and wait for a task during CPU fault handling
Reserved for future APU options	3-15	--

Active task name or waiting task name	is an 8-byte field containing the name of the currently active task. If the APU is stopped and waiting for a task, this field contains the name of the currently waiting task. The task name is left-justified with trailing blanks. If no currently active or waiting task exists, the entire field is filled with blanks.
Control task name	is an 8-byte field containing the name of the task having control rights over the specified APU. If no control task exists, the entire field is filled with blanks.
Mapping task name	is an 8-byte field containing the name of the task having mapping rights over the specified APU. If no mapping task exists, the entire field is filled with blanks.

Ready task name (n) or Exclusive task name	is a variable length table of 8-byte fields containing the name of each task in the ready queue of the specified APU. The order of entries in this table corresponds to the order of the tasks on the APU ready queue. When the specified APU has been marked on exclusively for one task, the ready queue is always empty. In this case, this field contains the name of the task having exclusive rights to the specified APU.

3.9.3 SVC 13 Code 2: Auxiliary Processing Unit (APU) Mapping

SVC 13 code 2 allows a task to perform mapping functions on a specified APU, provided the task has the mapping rights to the specified APU. A task is granted mapping rights to an APU only if:

- the requesting task has been Link edited with the APMAPPING option, and
- no other task has been granted mapping rights to that APU. Operator commands for APU mapping will not be accepted if a task already has these mapping rights.

Once a task has been granted mapping rights to an APU, the task can:

- mark the APU on,
- map the APU into the LPMT,
- remove all references to the APU from the LPMT, and
- mark the APU off.

Figure 3-17 shows the parameter block format and coding for SVC 13 code 2.

```
 ----------------------------------------------------------------
|0(0)          |1(1)           |2(2)                |3(3)         |
|   Options    | Function code |  Directive option  | APU number  |
|  (SV13.OPT)  |  (SV13.FUN)   |     (SV13.DOP)     | (SV13.APN)  |
|----------------------------------------------------------------|
|4(4)                          |6(6)         Error                |
|            Reserved          |          status code             |
|           (SV13.APS)         |           (SV13.ERR)             |
|----------------------------------------------------------------|
| 8(8)                                                           |
|                   Data buffer start address                    |
|                          (SV13.BUF)                            |
|----------------------------------------------------------------|
|12(C)                         |14(E)                             |
|            Reserved          |         Length of buffer         |
|           (SV13.USE)         |           (SV13.LEN)             |
 ----------------------------------------------------------------
```

```
            SVC    13, parblk
             .
             .
             .
            ALIGN 4
   parblk   DC     X'option(s)'
            DB     2
            DC     'LPU number' or 0
            DC     'APU number'
            DC     H'0'
            DS     2
            DC     A(BUFFER)
            DC     H'0'
            DC     H'number of bytes in buffer'
```

Figure 3-17 SVC 13 Code 2 Parameter Block Format and Coding

This parameter block must be 16 bytes long, fullword boundary aligned, and located in a task's writable segment. A general description of the fields in this parameter block follows.

Fields:

Options (SV13.OPT)	is a 1-byte field containing a hexadecimal number specifying one or more of the following mapping options:

OPTION CODES	FUNCTION
X'80'	Gain mapping rights for specified APU.
X'40'	Mark APU on, exclusive to only one task.
X'20'	Mark APU on (tasks can be added to APU queue).
X'10'	Map APU into LPMT at LPUn. (n is indicated by the directive option field at SV13.DOP.) If APU = 0, LPUn is mapped to the CPU.
X'08'	Remove all references to the specified APU from the LPMT.
X'02'	Mark APU off (removes all tasks from APU and APU queue).
X'01'	Release task's mapping rights for the specified APU.

NOTE

If two or more of the above option bits are specified, they are executed in a left-to-right order.

Function code (SV13.FUN)	is a 1-byte field that must contain the decimal number 2 to indicate code 2 of SVC 13.
Directive option (SV13.DOP)	is a 1-byte field that contains the LPU number for option X'10'. All other options ignore this field.
APU number (SV13.APN)	is a 1-byte field that must contain the number of the APU to which this SVC is directed.
Reserved (SV13.APS)	is an unused 2-byte field that should be initialized to zero (0).
Error status code (SV13.ERR)	is a 2-byte field that receives the execution status of SVC 13 code 1. The first byte of this field indicates the bit position (0-7) of the SVC 13 code 2 option being executed when the error occurred. The second byte contains one of the SVC 13 error status codes. See Table 3-10. If no error occurs, both bytes contain 0.

Data buffer start address (SV13.BUF)	is a 4-byte field that specifies the address of the buffer containing the name of the task to be granted exclusive access to the specified APU. The task name specified in this buffer must be 8 bytes long and left-justified with trailing blanks. If the entire buffer is filled with blanks, the task issuing the SVC is granted exclusive access to the specified APU. This field applies to option X'40' only; all other options ignore this field.
Reserved (SV13.USE)	is a 2-byte unused field that should be initialized to zero.
Length of buffer (SV13.LEN)	is a 2-byte field indicating the length of the data buffer containing the name of the task to be granted exclusive access to the specified APU. This field applies to option X'40' only; all other options ignore this field.

3.9.4 SVC 13 Code 3: Auxiliary Processing Unit (APU) Control

SVC 13 code 3 allows a task to perform control functions on a specified APU provided the task has obtained the control rights to the specified APU. OS/32 grants APU control rights to a requesting task only if:

- the task has been Link edited with the APCONTROL option, and
- no other task has been granted control privileges to the specified APU. Operator commands for APU control will not be accepted if a task already has these control rights.

SVC 13 code 3 gives a task the ability to:

- Initialize an APU (perform a power up link check).
- Send a directive to control APU task execution.
- Stop the APU and preempt the currently executing task.
- Preempt the next ready task on the APU ready queue with another task selected from the queue.
- Disable an APU for online maintenance.

Figure 3-18 shows the parameter block format and coding for SVC 13 code 3.

```
 ---------------------------------------------------------------
|0(0) APU         |1(1)           |2(2)               |3(3)        |
|control options| Function code | Directive option | APU number |
|  (SV13.OPT)    |  (SV13.FUN)   |    (SV13.DOP)     | (SV13.APN) |
|---------------------------------------------------------------|
|4(4)                            |6(6)                           |
|           Reserved             |       Error status code       |
|          (SV13.APS)            |          (SV13.ERR)           |
|---------------------------------------------------------------|
|8(8)                                                            |
|                  Data buffer start address                    |
|                         (SV13.BUF)                            |
|---------------------------------------------------------------|
|12(C)                           |14(E)                          |
|           Reserved             |        Length of buffer       |
|          (SV13.USE)            |          (SV13.LEN)           |
 ---------------------------------------------------------------
```

```
         SVC    13,parblk
          .
          .
          .
         ALIGN 4
parblk   DB     X'option'
         DB     3
         DB     X'directive option'
         DB     'APU number'
         DS     4
         DC     A(BUFFER)
         DC     H'0'
         DC     H'length of buffer'
```

Figure 3-18 SVC 13 Code 3 Parameter Block Format and Coding

This parameter block must be 16 bytes long, fullword boundary aligned, and located in a task writable segment. A general description of each field in the parameter block follows.

Fields:

APU control options (SV13.OPT)	is a 1-byte field specifying a hexadecimal number indicating the APU control option to be executed. Figure 3-19 shows the APU control option field format. See Table 3-6 for the available options for this field. If more than one APU control option is specified, the options are executed in a left-to-right order.

Function code (SV13.FUN)	is a 1-byte field that must contain the decimal number 3 to indicate code 3 of SVC 13.
Directive option (SV13.DOP)	is a 1-byte field specifying a hexadecimal command code to be sent to the specified APU. See Table 3-7. This field is used only if X'08' was specified in the APU control options field. The directive option field is ignored for all other APU control options.
APU number (SV13.APN)	is a 1-byte field (one through the number of APUs in the system) that identifies the specified APU. (APU 0 has no meaning here. By default all tasks have mapping and control rights to APU 0 but no other control functions are valid for APU 0).
APU hardware status (SV13.APS)	is a 2-byte field that will contain one of the following: • If option X'08' is specified and any command other than link check (X'80') is specified in the SV13.DOP field, this field receives the APU response status returned after execution of the specified byte. • If option X'08' is specified and the link check command (X'80') is specified in the directive option (SV13.DOP) field, the right-most byte of the halfword defines a data pattern (determined by the user), which is sent to the APU. The APU complements the byte and sends it back to the left byte of the field.
Error status code (SV13.ERR)	is a 2-byte field that receives the execution status of SVC 13 code 3. The first byte of this field indicates the bit position of the option being executed when the error occurred. The second byte of this field contains one of the SVC 13 error status codes. See Table 3-10 for a list of the SVC 13 error codes.
Data buffer start address (SV13.LEN)	is a 4-byte field that specifies the address of a buffer containing the name of the task on the APU ready queue that is to be selected as the next task to be executed. This task must be an existing member of the queue. This field applies to option X'10' only and is ignored for all other APU control options.

Reserved (SV13.USE)	is an unused 2-byte reserved field that should be initialized to zero.
Length of buffer (SV13.LEN)	is a 2-byte field specifying a decimal number (8 or greater) indicating the length of the data buffer specified by the SV13.BUF field. This field applies to option X'10' only and is ignored for all other APU control options.

	Gain control rights	Enable APU	Stop and reschedule APU task	Preempt task queue	Send APU command	Reserved	Disable APU	Release control rights
Bits:	0	1	2	3	4	5	6	7

Figure 3-19 SVC 13 APU Control Options Field (SV13.OPT)

TABLE 3-6 SVC 13 CODE 3 APU CONTROL OPTIONS (SV13.OPT) FIELD BIT DEFINITIONS

APU CONTROL PRIVILEGE OPTION	BIT POSITION	HEX CODE	DESCRIPTION	PREREQUISITES
Gain control rights	0	X'80'	Task gains control rights to the specified APU.	Task link edited with APCONTROL task option. No other task has control rights to specified APU.
Enable APU	1	X'40'	Initializes specified APU by performing a power up link check. After power up link check, APU is in an OFF and IDLE state.	APU must be in disabled state.

TABLE 3-6 SVC 13 CODE 3 APU CONTROL OPTIONS (SV13.OPT) FIELD BIT DEFINITIONS (Continued)

APU CONTROL PRIVILEGE OPTION	BIT POSITION	HEX CODE	DESCRIPTION	PREREQUISITES
Stop and preempt APU task	2	X'20'	Stops execution of the current task on the specified APU, saves the task context, and reschedules the current task to the rear of the APU ready queue. APU is left in IDLE state.	APU must be enabled. APU must have a currently exe-cuting task.
Select next task	3	X'10'	Select the task specified in the buffer from the APU ready queue as the next task for the APU to run. Tasks appearing in the APU ready queue after this task will execute in order.	APU must be enabled.
Send APU Command	4	X'08'	Send the APU command specified in the SVC directive option field (SV13.DOP) to the specified APU.	See Table 3-7.
Reserved	5	X'04'	Reserved for future use.	
Disable APU	6	X'02'	Disable the specified APU.	APU must be in OFF state.
Release control rights	7	X'01'	Task gives up con-trol rights to the specified APU.	None.

TABLE 3-7 SVC 13 CODE 3 APU COMMANDS (SV13.DOP)

HEX CODE	MEANING	PREREQUISITES
X'01'	Start APU for task execution. APU enters running state.	APU must be in IDLE state.
X'02'	Execute single instruction.	APU must be in IDLE state. Reserved for diagnostic use.
X'04'	Transfer current task to CPU Not recommended for user-written tasks. See SVC 6 example in Chapter 6.	APU must be in IDLE state. APU must have a current task.
X'07'	Start APU for nontask execution - loads and starts APU using power fail image.	APU must be in IDLE state. Reserved for diagnostic use.
X'08'	Store power fail image	APU must be in IDLE state. Reserved for diagnostic use.
X'0B'	Stop APU if task state - saves context of current executing task and stops APU (APU enters IDLE state.)	APU must be running in task state (PSW 15=0). APU must have a current task.
X'80'	RTSM link check - sends data byte and receives complement.	APU must be in IDLE state. Reserved for diagnostic use.
X'83'	Reschedules task on APU - reschedules the current task to the rear of the APU ready queue.	APU must have a current task. APU must be in IDLE state.

NOTE

The preferred method of rescheduling a task on an APU is to issue an SVC 13 code 3 with the X'20' APU control option specified.

TABLE 3-7 SVC 13 CODE 3 APU COMMANDS (SV13.DOP) (Continued)

HEX CODE	MEANING	PREREQUISITES
X'85'	Stop APU and save power fail image saves context of currently executing task and stops APU (APU enters IDLE state).	Use only after attempt to stop APU with X'0B' has failed due to APU command sequence error.
X'86'	Fetches APU status.	APU must not be disabled.
X'89'	Fetch APU error code.	Reserved for diagnostic use.
X'8A'	Checkpoint task state saves context of current task. Task continues execution if previously running or remains idle if previously idle.	APU must have a current task.

NOTE

If an undefined command code in the SV13.DOP field is to be sent to the APU, the APU will identify the code sent as an unrecognizable command code or as a sequence error. For more information on the SVC 13 APU commands, see the Model 3200MPS System Instruction Set Reference Manual.

3.9.5 SVC 13 Auxiliary Processing Unit (APU) Hardware Status (SV13.APS) Field

After execution of SVC 13 code 1 or SVC 13 code 3 option X'08', the status of the APU hardware is saved in the APU status (SV13.APS) field of the parameter block. This field consists of two bytes, a response byte and an error code byte, representing the response and error fields of the PSW. The hardware status is returned to this field in the format shown in Figure 3-20. See Table 3-8 for the response byte bit definitions. Error codes returned to the error code byte are listed in Table 3-9. See the Model 3200MPS System Instruction Set Reference Manual for more information.

```
            RESPONSE BYTE                      ERROR CODE BYTE
------------------------------------------------------------------
| P | R |       | W | R | E | M | M |             |              |
|   |   | NON-  | A | E | R | O | O |             |              |
| A | U |       | I | S | R | D | D |    ERROR    |     CODE     |
|   |   | TASK  | T | P | O | 1 | 2 |             |              |
| R | N |       |   |   | R |   |   |             |              |
------------------------------------------------------------------
Bits:
 0                                 7 8                          15
```

Figure 3-20 APU Hardware Response Byte (SV13.APS)

TABLE 3-8 APU HARDWARE RESPONSE BYTE BIT DEFINITIONS

BIT POSITION	BIT NAME	BIT SETTING AND MEANING
0	PAR	1= ensures that the response byte has an odd parity 0= odd number of bits have been set for the remainder of the byte
1	RUN	1= APU is running 0= APU is idle
2	NONTASK	1= current PSW bit 15 is set, indicating no context save area is available 0= APU executing in task state; the current PSW bit 15 is not set, indicating that the current task's context save area is ready to receive the processor task state
3	WAIT	1= current PSW bit 16 is set, indicating APU is in a wait state or APU is working in an internal service state (e.g., scheduling a task) 0= current PSW bit 16 is not set, indicating the APU is executing instructions
4	RESP	1= APU is responding to a command from the CPU 0= APU is generating a signal indicating a change in APU state

TABLE 3-8 APU HARDWARE RESPONSE BYTE BIT DEFINITIONS (Continued)

BIT POSITION	BIT NAME	BIT SETTING AND MEANING
5	ERROR	1= APU detects an error condition that causes the APU to stop; the error condition is indicated by the setting of the error code bit (see Table 3-9) 0= no error condition is detected by APU
6,7	MOD1 MOD2	Bit definitions for MOD1 and MOD2 depend on the definitions for RESP and ERROR, as follows: RESP=0, ERROR=0 00 = undefined 01 = APU entering queue wait state 10 = task rescheduled to APU ready queue 11 = task rescheduled to CPU RESP=0, ERROR=1 00 = general error status 01 = error occurred while APU in queue wait state 10 = error occurred while locking queue 11 = undefined RESP=1, ERROR=0 00 = general response status 01 = task is waiting on APU queue 10 = APU attempting to lock a queue 11 = command sequence error; command was ignored RESP=1, ERROR=1 00 = error as a result of command 01 = response, error in queue wait 10 = response, error in queue lock 11 = error and command sequence error

NOTE

See the Model 3200MPS System Instruction Set Reference Manual for more information on the nontask and wait states that reflect the PSW bit definitions.

TABLE 3-9 ERROR CODES FOR ERROR CODE BYTE OF APU HARDWARE STATUS (SV13.APS) FIELD

ERROR CODE	MEANING
X'80'	No error
X'01'	APUID DEVICE FALSE SYNC
X'02'	ZERO APUID RETURNED BY RTSM
X'83'	CAN'T FETCH WORD @ X'C4' - ECC
X'04'	APUID > MAX APU @ X'C7'
X'85'	BAD A(APB_DIR) - ECC/ZERO/ALIGN
X'86'	BAD A(APB) - ECC/ZERO/ALIGN
X'07'	BAD APB (FLAGS:APB#) WORD - ECC
X'08'	WRONG APB# IN APB
X'89'	APB PASSBACK
X'8A'	UNRECOGNIZED COMMAND
X'0B'	BAD APB A(CTCB) - ECC/ZERO/ALIGN
X'8C'	BAD A (APU QUEUE) - ECC/ZERO/ALIGN
X'0D'	QUEUE LOCK TIMEOUT
X'0E'	EXECUTION SUSPENDED (TRAP PSW WAIT)
X'8P'	BAD SSTD - ECC
X'10'	CAN'T LOAD TASK CONTEXT
X'91'	CAN'T STORE TASK CONTEXT
X'92'	CAN'T LOAD PWR FAIL IMAGE
X'13'	CAN'T STORE POWER FAIL IMAGE
X'94'	CAN'T LOAD PSTD - ECC
X'15'	BAD APB PFAIL PTR - ECC/ZERO
X'16'	BAD APB MMF NEW PSW - PCC/ZERO
X'97'	BAD CTCB CTX PTR - ECC/ZERO/ALIGN
X'98'	BAD APB TCB CNT WORK - ECC
X'19'	BAD A(APU FRONT TCB) - ECC/ZERO/ALIGN
X'1A'	FRONT TCB PTR< TCB CNT DISAGREE
X'9B'	QUEUE TCB CNT UNDERFLOW
X'1C'	BAD APB A(CPU QUEUE) - ECC/ZERO/ALIGN
X'9D'	BAD TCB QHPTR - ECC/ZERO/ALIGN
X'9E'	INCORRECT TCB QUEUE HEAD PTR
X'1F'	BAD TCB BPTR - ECC/ZERO/ALIGN

TABLE 3-9 ERROR CODES FOR ERROR CODE BYTE OF APU HARDWARE STATUS (SV13.APS) FIELD (Continued)

ERROR CODE	MEANING
X'20'	BAD BACK TCB FPTR - ECC/ZERO/ALIGN
X'A1'	BACK TCB FPTR NOT TO FRONT TCB
X'A2'	BAD FRONT TCB FPTR - ECC/ZERO/ALIGN
X'23'	BAD FWD TCB BPTR - ECC/ZERO/ALIGN
X'A4'	FWD TCB BPTR NOT TO FRONT TCB
X'25'	INCONSISTENT FRONT TCB FPTR & BPTR
X'26'	BAD FRONT TCB PTR - ECC/ZERO/ALIGN
X'A7'	BAD BACK TCB FPTR - ECC/ZERO/ALIGN
X'A8'	BACK TCB'S FPTR NOT TO FRONT TCB
X'29'	TCB QUEUE OVERFLOW (CPU OR APU)
X'2A'	BAD MSH TIME ACCUMULATOR - ECC
X'AB'	BAD LSH TIME ACCUMULATOR - ECC
X'2C'	BAD TCB START TIME WORD - ECC
X'AD'	CAN'T READ RTSM CLOCK DATA
X'AE'	TCB ELAPSED TIME OVERFLOW
X'2F'	TCB "PENDING" FLAGS SET ON QUEUE OR CTCB
X'B0'	BAD "PENDING" FLAGS WORD - ECC
X'31'	INTERRUPT FROM RTSM XMTR
X'32'	CAN'T LOAD PFAIL PSTD - ECC
X'B3'	CAN'T LOAD PFAIL PSTD - ECC
X'B4'	BAD APB MSH TIME ACC - ECC
X'B5'	BAD APB LSH TIME ACC - ECC
X'B6'	WRONG APU NUMBER IN SBC 'FLAGS' WORD

3.9.6 SVC 13 Error Status Code (SV13.ERR) Field

When execution of an SVC 13 is completed, the execution status is returned to the error status code (SV13.ERR) field of the parameter block. If no error occurs, a value of 0 is stored in this field. If SVC 13 code 2 or code 3 is issued and an error occurs, the first byte of this field contains the bit position of the option that caused the error.

Table 3-10 lists the SVC 13 error status codes and their applicable function codes.

TABLE 3-10 SVC 13 ERROR STATUS CODES (SV13.ERR)

STATUS CODE	APPLICABLE FUNCTION CODES	MEANING
0	all	No errors occurred.
1	all	The specified data buffer does not begin on a fullword boundary.
2	0,1	The specified data buffer is not located in a writable segment of the task.
3	all	Insufficient space was available in the supplied data buffer. For functions 0 and 1, any data that does not fit in the available space is lost.
4	2,3	Task establishment options prohibit the task from gaining mapping or control rights.
5	2,3	Task has not been granted the rights to perform the attempted mapping or control function.
6	1,2,3	The APU number specified is greater than the maximum allowed.
7	2 option X'10'	The LPU number specified is greater than the maximum allowed.
8	2,3	An invalid option was specified for this function.
9	2,3	The requested privilege is currently held by another task and cannot be granted.
10	2 option X'40'	The specified APU cannot be marked on exclusive from ON state.

TABLE 3-10 SVC 13 ERROR STATUS CODES (SV13.ERR)
(Continued)

STATUS CODE	APPLICABLE FUNCTION CODES	MEANING
11	2,3	The function requested cannot be completed because the specified APU is in DISABLED state.
12	1,2,3	Access to the task queue for the specified APU could not be obtained; SVC 13 request was aborted.
13	2 option X'10'	The task has not been granted mapping rights over the APU currently mapped to the specified LPU.
14	3 option X'40'	Cannot enable APU unless in DISABLED state.
15	3 option X'40'	APU could not pass power up link check sequence. APU left in DISABLED state.
16	3 option X'02'	Cannot disable APU unless in an OFF state.
17	2 option X'40'	The APU could not be marked on exclusive because the specified task could not be found in the system.
18	3 option X'08'	Error encountered in transmission of the specified control command.
19	3 option X'10'	The preemptive task could not be found on the specified APU ready queue. The APU queue is unchanged.

3.9.7 Typical Option Coding Sequences for SVC 13 Code 2 and Code 3

The options field (SV13.OPT) in the SVC 13 parameter blocks for codes 2 and 3 allows the user to issue one call to execute multiple APU mapping or control functions. Multiple options are executed from left-to-right. Care must be taken when selecting the sequence of options to perform a designated APU control or mapping procedure. The following sections demonstrate specific option coding sequences that would be used by a typical APU control task in a Model 3200MPS System.

3.9.7.1 Auxiliary Processing Unit (APU) Initialization and Start Up

Before a task can run on an APU, the APU must be initialized and started for task execution. This is accomplished through an SVC 13 code 3 with the following sequence of option codes specified:

OPTION CODES	FUNCTIONS PERFORMED
X'80'	Gain control rights for task
X'40'	Enable (initialize) APU
X'08'	Send APU start directive (X'01') specified in SV13.DOP field
X'01'	Release control rights

3.9.7.2 Auxiliary Processing Unit (APU) Mark On

After initialization, the APU must be mapped into the LPMT and assigned to the LPU number to which the task that will run on the APU is directed. This is accomplished through an SVC 13 code 2 with the following sequence of option codes specified:

OPTION CODES	FUNCTIONS PERFORMED
X'80'	Gain mapping rights for task
X'20'	Mark APU on
X'10'	Map APU into LPMT
X'01'	Release mapping rights

The SV13.DOP field contains the LPU number to which the specified APU (SV13.APN) is to be mapped.

If the APU is to be marked on exclusively for one task, the following option coding sequence is used for SVC 13 code 2:

OPTION CODES	FUNCTIONS PERFORMED
X'80'	Gain mapping rights for task
X'40'	Mark APU on, exclusive
X'10'	Map APU into LPMT
X'01'	Release mapping rights

The buffer, identified by the address specified in the SV13.BUF parameter block field, contains the name of the task for which exclusive access is requested. SV13.DOP contains the LPU number to which the specified APU (SV13.APN) is to be mapped.

3.9.7.3 Effective Task Scheduling on the Auxiliary Processing Unit (APU) Queue

Scheduling is initially done by the task manager; the preempting process changes normal scheduling of tasks on an APU. Two methods are available for preemptive scheduling of the next ready task on the APU ready queue:

- Stop the APU, reschedule the currently active task to the rear of the queue, and restart the APU, which will automatically select the next task from the front of the queue.
- Stop the APU, reschedule the currently active task to the rear of the queue, assign another task on the queue to the front of the queue, and start the APU, which will automatically select it as the next task to execute.

To schedule the current task to the rear of the queue, use SVC 13 code 3 with the following sequence of option codes specified:

OPTION CODES	FUNCTIONS PERFORMED
X'80'	Gain control rights for task
X'20'	Stop APU task execution and reschedule current task to rear of the queue

X'08'	Send APU start directive (X'01') specified in SV13.DOP field to select the task at the front of the APU ready queue for execution
X'01'	Release control rights

To preempt the next ready task, thereby explicitly selecting the next task to be run, use SVC 13 code 3 with the following sequence of option codes specified:

OPTION CODES	FUNCTIONS PERFORMED
X'80'	Gain control rights for task
X'20'	Stop APU task execution and reschedule current task to rear of the queue
X'10'	Change front of queue pointer to designated task (name of task is specified in the buffer identified by the address in the SV13.BUF field)
X'08'	Send APU start directive (X'01') specified in SV13.DOP field to start task
X'01'	Release control rights

3.9.7.4 Auxiliary Processing Unit (APU) Mark Off

An APU can be marked off with or without changing its LPU to APU mapping assignments in the LPMT. To mark off an APU, use SVC 13 code 2 with the following sequence of option codes specified:

OPTION CODES	FUNCTIONS PERFORMED
X'80'	Gain mapping rights for task
X'02'	Mark APU off
X'01'	Release mapping rights

To mark off an APU and remove all of its references from the LPMT (i.e., remap all LPUs currently assigned to the APU to the CPU), use SVC 13 code 2 with the following sequence of option codes specified:

OPTION CODES	FUNCTIONS PERFORMED
X'80'	Gain mapping rights for task
X'08'	Remove all references to APU number from LPMT, no matter what LPUs are assigned to it
X'02'	Mark APU off
X'01'	Release mapping rights

CHAPTER 4
SUPERVISOR CALL (SVC) INTERCEPTION

4.1 INTRODUCTION

SVC interception software is used to write programs that can emulate the SVC processing ability of OS/32. This software consists of macros that allow a task (intercepting task) to intercept the SVC of another task before it goes to the operating system for processing. Once intercepted, the SVC can be monitored by the intercepting task and sent to the operating system for processing, or it can be processed by the intercepting task. Table 4-1 lists the system macros used for SVC interception.

TABLE 4-1 SYSTEM MACROS FOR SVC INTERCEPTION

MACRO	FUNCTION
ICREATE	Creates an SVC intercept path.
IREMOVE	Removes a previously created path.
IGET	Gets data from a data area of the task that issued an intercepted SVC.
IPUT	Puts data into a data area of the task that issued an intercepted SVC.
ICONT	Continues standard execution of an intercepted SVC by passing control to an OS/32 SVC executor.
IPROCEED	Allows the task that issued the intercepted SVC to proceed with its execution.
IROLL	Makes an intercepted task rollable.
ITERM	Terminates an intercepted SVC after processing.
ITRAP	Sends a task queue trap to a task.
IERRTST	Evaluates errors returned by any of the above macros and branches execution to specific error routines within the intercepting task.

The intercepting task tells the OS/32 SVC executor which SVC it will process or monitor. When the intercepting task is sent an SVC from the executor, the intercepting task handles the intercepted SVC while the task that issued the SVC is placed in a wait state. While executing the intercepted SVC, the intercepting task can read from or write to the address space of the task that issued the SVC.

A task is not aware that its SVC has been intercepted unless it is informed by the intercepting task.

SVC interception software must be configured in OS/32 at the time of system generation (sysgen). See the INTERCEPT configuration statement in the OS/32 System Generation (SYSGEN/32) Reference Manual.

A task can intercept SVC calls only after it is linked with the intercept task option enabled (OPTION INTERCEPT). See the OS/32 Link Reference Manual. The task can then be programmed to intercept any of the following SVCs issued by any application task in the system:

- SVC 1
- SVC 2 code 7
- SVC 3
- SVC 6
- SVC 7

Intercepting tasks can be loaded and executed under MTM. However, the intercepting task must be loaded from an account that has executive task (e-task) load privileges. See the OS/32 Multi-Terminal Monitor (MTM) System Planning and Operation Reference Manual.

4.2 HOW SUPERVISOR CALL (SVC) INTERCEPTION WORKS

In general, SVC interception software functions as follows:

1. A task with SVC interception enabled by Link is built. This intercepting task must:

    - reserve memory for a set of request descriptor block (RDB) buffers for each SVC to be intercepted,
    - build a circular list for storing addresses of RDB buffers containing information on intercepted SVCs,
    - create (via the ICREATE macro) intercept paths that designate the SVCs to be intercepted, and
    - define (via the ICREATE macro) what control the intercepting task has over the SVCs it intercepts.

2. An application task issues an SVC.
3. If no intercept path was created for that particular SVC, one of the standard OS/32 executors services the SVC.
4. If an intercept path has been created for that SVC, the operating system:

    - intercepts the SVC before it reaches the OS/32 executor,
    - removes an RDB address from the circular list of the intercepting task,
    - loads the SVC's parameter block and identifying information into the RDB, and
    - sends a task event trap to the intercepting task to notify the task that an SVC has been intercepted.

5. Execution of the intercepting task branches to the task event trap handling routine. The address of this routine is specified when the path is created via the ICREATE macro.

6. If the intercept path was built to monitor this SVC, the task event trap handling routine issues an ICONT macro to return the SVC to the OS/32 executor for execution.

7. If the intercept path was built to service the SVC, the task event trap handling routine processes the SVC by the intercept macros IGET, IPUT, IROLL, and ITRAP. Also, the routine can issue the IPROCEED macro to allow the application task to continue executing during SVC processing.

8. After the task event trap handling routine processes the SVC, it issues an ITERM macro that transfers control back to the application task that issued the SVC.

9. The intercepting task exits the trap handler through the TEXIT macro.

4.3 PREPARING A TASK FOR SUPERVISOR CALL (SVC) INTERCEPTION

Before creating an intercept path, an intercepting task must:

- build a set of RDB buffers for each type of SVC to be intercepted,
- build a circular list to store the addresses of the RDB buffers, and
- be prepared to handle a task event trap.

4.3.1 Request Descriptor Block (RDB) Buffers

The size of each RDB buffer built by the intercepting task depends on the size of the parameter block for the particular SVC that is to be intercepted. For example, a set of buffers allocated for SVC 6 interception will be larger than a set of buffers for SVC 1 interception. When an intercepting task uses one set of buffers for intercepting two or more SVC types, the buffer size must equal the size of the RDB needed to hold the largest parameter block associated with the SVCs to be intercepted. Figure 4-1 shows the RDB fields. To define a structure containing these fields, use the $RDB macro.

```
 -----------------------------------------------------------
|0(00)                        |2(02)   Intercept path        |
|         Reserved            |          identifier          |
|        (RDB.RID)            |          (RDB.PID)           |
|------------------------------------------------------------|
|4(04) Parameter block offset |6(06) SVC     |7(07)  Task    |
|          (RDB.OFF)          |    type      |   priority    |
|                             |  (RDB.SVC)   |  (RDB.TPRI)   |
|------------------------------------------------------------|
|8(08)                                                       |
|              Operating system task identifier              |
|                         (RDB.TID)                          |
|------------------------------------------------------------|
|12(0C)                                                      |
|                SVC parameter block address                 |
|                         (RDB.PAD)                          |
|------------------------------------------------------------|
|16(10)           Instruction address following              |
|                  intercepted SVC instruction               |
|                          (RDB.SVAD)                        |
|------------------------------------------------------------|
|20(14)                                                      |
|                     SVC parameter block                    |
|                          (RDB.PB)                          |
|                             .                              |
|                             .                              |
|                             .                              |
|                                                            |
|                                                            |
|                                                            |
 -----------------------------------------------------------
```

Figure 4-1 Request Descriptor Block

The fields contained within the RDB are described as follows.

Fields:

Reserved (RDB.RID)	is a halfword field reserved for future use.
Intercept path identifier (RDB.PID)	is a halfword field containing an SVC intercept path identifier exclusively reserved for one particular SVC interception.
Parameter block offset (RDB.OFF)	is a halfword field containing the hexadecimal offset value for the parameter block field within the RDB.

SVC type (RDB.SVC)	is a 1-byte field containing a decimal number specifying the type of SVC that is to be intercepted. • 01 indicates SVC 1 • 02 indicates SVC 2 code 7 • 03 indicates SVC 3 • 06 indicates SVC 6 • 07 indicates SVC 7
Task priority (RDB.TPRI)	is a 1-byte field containing a decimal number specifying the priority of the task that issued the intercepted SVC.
Operating system task identifier (RDB.TID)	is a 4-byte field containing the operating system task identifier for the task that issued the intercepted SVC.
SVC parameter block address (RDB.PAD)	is a 4-byte field containing a hexadecimal number specifying the address of the parameter block for the SVC being intercepted. For SVC 3 interceptions, this field contains the end of task code.
Instruction address following intercepted SVC instruction (RDB.SVAD)	is a 4-byte field containing a hexadecimal number specifying the address of the instruction following the intercepted SVC instruction. This field is set to 0 for SVC 3 interceptions.
SVC parameter block (RDB.PB)	is a variable length field containing the parameter block of the intercepted SVC.

4.3.2 Circular List for Request Descriptor Block (RDB) Buffers

The intercepting task must have a standard Perkin-Elmer circular list to hold the address of each RDB buffer. Figure 4-2 shows the fields of the standard circular list. When an SVC is sent to the intercepting task for processing, one RDB buffer address is automatically removed from the circular list, and the RDB is filled with information identifying the intercepted SVC. The circular list can be created by the assembler instruction DLIST. Refer to the appropriate Perkin-Elmer Series 3200 Processor User's Manual for a more detailed explanation of the standard circular list.

```
 ------------------------------------------------------------
|0(00)                         |2(02)                          |
|       Maximum number         |        Current number         |    |
|        of buffers            |         of buffers            |
|------------------------------------------------------------|
|4(04)                         |6(06)                          |
|         Current top          |        Next bottom            |
|                              |                               |
|------------------------------------------------------------|
|8(08)                                                         |
|                       A (buffer 1)                           |
|                                                              |
|------------------------------------------------------------|
|12(0C)                                                        |
|                       A (buffer 2)                           |
|                                                              |
|------------------------------------------------------------|
|                                                              |
|                            .                                 |
|                            .                                 |
|                            .                                 |
|                                                              |
|------------------------------------------------------------|
|                                                              |
|                       A (buffer n)                           |
|                                                              |
 ------------------------------------------------------------
```

Figure 4-2 System Task Buffer List (Standard Circular List)

Fields:

Maximum number of buffers	is a halfword field indicating the maximum number of fullwords in the entire list. |
Current number of buffers	is a halfword field indicating the number of fullwords currently in use. When this field equals 0, the list is empty. When this field equals the number of fullwords in the list, the list is full.
Current top	is a halfword field indicating the address of the RDB buffer that is currently at the top of the list.
Next bottom	is a halfword field indicating the address of the next RDB buffer that is at the bottom of the list.
A (buffer n)	indicates the address of an RDB buffer.

4.3.3 Task Event Trap

To receive a task event trap, an intercepting task must have the TSW.TESB bit in its task status word (TSW) set. See the OS/32 Application Level Programmer Reference Manual for more information on TSW bit settings. If this bit is not set, the task event trap will be queued until a TSW is loaded with this bit set. In addition, a task cannot receive a task event trap or task queue trap during execution of the task event trap handling routine. These traps will be queued until the task exits from the routine.

Before execution branches to the task event trap handling routine, the operating system places the address of the RDB in register 1 and a unique intercept path identifier in register 0. To prevent the data in these registers from being lost during execution of the task event trap handling routine, the intercepting task should be link edited with the TEQSAVE task option. TEQSAVE informs the operating system which register contents should be saved and restored when a task enters or exits the task event trap handling routine. See the OS/32 Link Reference Manual for more information on TEQSAVE.

4.4 CREATING INTERCEPT PATHS (ICREATE)

Before an intercepting task can intercept an SVC, it must create a path to the application task that contains the SVC to be intercepted. This path is created by executing code built by the ICREATE macro that informs the OS/32 SVC executor which SVC is to be intercepted by this path. The intercepting task also accesses the application task's address space through the intercept path.

An intercept path remains in effect until it is removed by the intercepting task creating it or until the intercepting task terminates. Although only one type of SVC can be intercepted by each path, there is no limit to the number of paths that can be created by one intercepting task.

The mode parameter of the ICREATE macro specifies when an SVC is to be intercepted. Under caller mode, the specified SVC is intercepted every time it is issued by the application task. When the recipient existent mode is specified, the SVC is intercepted only when it is directed towards a specified task, device, pseudo task, or pseudo device that exists in the system. Under the recipient nonexistent mode, the SVC is intercepted only when it is directed toward a specified pseudo task or pseudo device created by execution of code built by the ICREATE macro.

4.5 HOW TO CREATE A PSEUDO DEVICE OR TASK WITH ICREATE

A pseudo device consists of a name and the SVC 1 or SVC 7 intercept paths attached to it. The pseudo device name, which is known to the system but does not actually refer to any system device or file, consists of a device name, filename, and extension. A device name that does not already exist for a real device or disk volume must be used. Pseudo devices ignore the file class/account number field of the file descriptor.

When the operating system cannot find a device or filename in the system, it will search the list of pseudo devices. If a match occurs, the system will continue processing the SVC using the pseudo device.

To create a pseudo device using SVC interception software, the ICREATE macro should be set to specify either an SVC 1 or SVC 7. The recipient nonexistent mode should also be specified. An SVC 1 intercept path must be in effect when an I/O operation is attempted to a pseudo device; otherwise, an invalid function (X'C0') error status is returned.

A pseudo task consists of a name attached to one or more SVC 6 intercept paths. A pseudo task name is known to the system but does not refer to an actual task existing in the system.

To create a pseudo task, issue the ICREATE macro specifying SVC 6 and the recipient nonexistent mode. Because a pseudo task does not refer to a real task, the pseudo task cannot be cancelled. Both pseudo tasks and pseudo devices can be deleted by removing all intercept paths attached to them.

4.6 USE OF GENERIC NAMING FOR PSEUDO DEVICES AND TASKS

A pseudo device or task can be generically named. The following characters can be used for generic naming:

- An asterisk (*) represents any character or blank.
- A backward slash (\) represents any character.

If a pseudo device or task name specifies the filename and extension fields as blanks, the system substitutes filename and extension fields filled with asterisks. This has the effect of generically naming the filename and extension fields so that they will always match the input filename and extension.

If the operands of an ICREATE macro specify the recipient existent mode and a generic pseudo device or task name, a pseudo device or task must exist with its name exactly matching the one specified by ICREATE. An error will result if the names do not match. For example, a system is asked to create the following pseudo devices:

- FAKE:FILE1
- FAKE:FILE*
- FAKE:
- FAKE:FILE*.EXT

Normally, the following input will match the above pseudo devices:

INPUT NAME	SELECTED PSEUDO DEVICE
FAKE:	FAKE:
FAKE:FILE3	FAKE:FILE*
FAKE:FILE1	FAKE:FILE1
FAKE:FILE11	FAKE:
FAKE:FILEX.EXT	FAKE:FILE*.EXT
FAKE:FILEX.EX	FAKE:

When the code built by the ICREATE macro is issued specifying recipient nonexistent mode and the pseudo device FAKE:, the ICREATE function will not be performed because the pseudo device already exists. Consequently, when an ICREATE macro is used specifying recipient existent mode along with the pseudo device FAKE:FILE*, ICREATE will be executed because the pseudo device FAKE:FILE* already exists.

4.7 FUNCTIONAL SUMMARY OF SUPERVISOR CALL (SVC) INTERCEPTION

The following describes how interception works for each SVC and mode:

SVC 1 caller	Any SVC 1 issued by the specified task is intercepted.
SVC 1 recipient existent	Any SVC 1 directed to an lu assigned to the specified device or pseudo device is intercepted. (Note that disk volume interception is not supported for SVC 1.)
SVC 1 recipient nonexistent	The pseudo device is created, and any SVC 1 call specifying an lu assigned to this pseudo device is intercepted.

SVC 2 code 7 caller	Any SVC 2 code 7 issued by the specified task is intercepted.
SVC 2 code 7 recipient existent	This call is invalid.
SVC 2 code 7 recipient nonexistent	This call is invalid.
SVC 3 caller	If the specified task goes to end of task for any reason, an SVC 3 intercept will occur.
SVC 3 recipient existent	This call is invalid.
SVC 3 recipient nonexistent	This call is invalid.
SVC 6 caller	Any SVC 6 issued by the specified task is intercepted.
SVC 6 recipient existent	Any SVC 6 directed to the specified task or pseudo task is intercepted.
SVC 6 recipient nonexistent	The pseudo task is created, and any SVC 6 call directed to this pseudo task is intercepted.
SVC 7 caller	Any SVC 7 issued by the specified task is intercepted.
SVC 7 recipient existent	Any SVC 7 directed to the specified device, disk volume, or pseudo device is intercepted.
SVC 7 recipient nonexistent	The pseudo device is created, and any SVC 7 call specifying this pseudo device is intercepted.

4.8 FULL AND MONITOR CONTROL INTERCEPT PATHS

The ICREATE macro specifies the level of control that the intercept path allows an intercepting task to have over an application task.

A full control intercept path allows the intercepting task to exert full control over a task whose SVC has been intercepted. Specifically, the intercepting task can:

Either ICONT or ITERM can be used to terminate interception from a monitor control intercept path. The system does not differentiate between the two calls in this case. Here the ICONT or ITERM macro replaces the RDB buffer address back on the circular list. It is very important that the ICONT or ITERM macro be used to replace the RDB.

Cancelling an application task under monitor or full control aborts the processing of the intercepted SVC in progress. The intercepting task must still issue an ICONT or ITERM to terminate the SVC interception.

4.11 HOW TO REMOVE INTERCEPT PATHS

An intercepting task can remove an intercept path by executing code built by an IREMOVE macro specifying the path to be removed. IREMOVE can be used for both immediate and delayed termination depending on whether the controlled shutdown or abort option is chosen.

The controlled shutdown option refuses all incoming requests and completes the servicing of all existing queued and executing SVCs. When processing of the last existing SVC intercepted by the path is completed, the path is removed from the system.

The abort option terminates all existing queued and executing SVCs before removing the intercept path from the system.

4.12 ERROR HANDLING

Run-time errors that result from executing intercept macro code are handled by user written error routines within the intercepting task. When an error occurs, execution branches to the routine specified by either the IERRTST macro statement or the error parameter associated with each macro.

The IERRTST macro is issued immediately after a macro for which the error parameter has been omitted. If an error occurs, execution of the intercepting task will branch to a user written error routine to handle the error. Error codes returned by the IERRTST macro are listed in Table 4-2. If no error occurs, execution continues at the instruction following the IERRTST macro.

If the ERROR parameter is specified with an intercept macro and an error occurs, execution branches to the specified error routine within the intercepting task. If no error occurs, execution proceeds to the next executable statement. The error routine pointed to by the ERROR parameter can contain an IERRTST macro to identify what error has occurred.

TABLE 4-2 ERROR CODES RETURNED FOR INTERCEPT MACROS

ERROR CODE	MEANING	RELEVANT MACROS
MO	Invalid interception mode	ICREATE
AD	Invalid address in parameter control block (PCB)	ICREATE ITERM ICONT IREMOVE ITRAP IGET IPUT
EX	Task or device exists when it should	ICREATE
SP	Insufficient system space to do request, or NINTC>64, or PBSIZE>998	ICREATE ITERM ITRAP IGET IPUT
CT	Full control already selected	ICREATE IROLL IPROCEED ITRAP IGET IPUT
HA	Invalid queue handler name	ICREATE
FD	Invalid device name or task name	ICREATE
ST	Invalid state for call; e.g., IROLL followed by ICONT or issuing IPUT with monitor control intercept path	ICONT IREMOVE IROLL IPROCEED ITRAP IGET IPUT
TP	Task queue item not added	ITRAP
RD	Invalid RDB	ITERM ICONT IROLL IPROCEED ITRAP IGET IPUT

TABLE 4-2 ERROR CODES RETURNED FOR INTERCEPT MACROS (Continued)

ERROR CODE	MEANING	RELEVANT MACROS
ID	Intercept path corresponding to this path ID does not exist	IREMOVE
WR	Attempt to copy SVC parameter block back into write protected area	ITERM
CD	Invalid subcode in SVC parameter block SVC interception software not included at sysgen	all
NT	Intercepted task has gone to end of task	IROLL IPROCEED ITRAP IGET IPUT

4.13 MACROS USED WITH SUPERVISOR CALL (SVC) INTERCEPTION

Once configured for SVC interception, the operating system allows tasks to execute code built by macros for SVC interception provided the tasks were linked with the intercept option.

This section gives the syntax for the SVC macros described in the previous sections. See the OS/32 System Macro Library Reference Manual for a list of syntax rules.

4.13.1 ICREATE Macro

The ICREATE macro creates an intercept path for a particular SVC type. See Table 4-3 for valid combinations for the SVC, MODE, and NAME parameters.

Format:

```
  NAME  |  OPERATION  |                 OPERAND
-----------------------------------------------------------
        |             |      ( (1)   )
        |             |      { (2,7) }
symbol  |  ICREATE    | SVC= { (3)   }
        |             |      { (6)   }
        |             |      ( (7)   )
        |             |
        |             |        (CL)
        |             | ,MODE= {RX}
        |             |        (RN)
        |             |
        |             | ,NAME=pointer
        |             |
        |             | ,TID=pointer
        |             |
        |             |           (FC)
        |             | ,CONTROL= {  }
        |             |           (MC)
        |             |
        |             | ,BUFFERL=pointer
        |             |
        |             | [,HANDLER=pointer]
        |             |
        |             | ,PID=pointer
        |             |
        |             | ,EXEC=pointer
        |             |
        |             | [,PBSIZE=n]
        |             |
        |             | [,SVAR=pointer]
        |             |
        |             | [,ERROR=pointer]
        |             |
        |             | [,PCB=pointer]
        |             |
        |             | [,FORM=L]
        |             |
        |             | [,NINTC=n]
        |             |
```

Parameters:

SVC=

is an integer, enclosed by parentheses, that indicates the type of intercept path to be created:

- (1) indicates SVC 1
- (2,7) indicates SVC 2 code 7
- (3) indicates SVC 3
- (6) indicates SVC 6
- (7) indicates SVC 7

MODE=

indicates one of the following interception modes:

- CL indicates caller mode
- RX indicates recipient existent mode
- RN indicates recipient nonexistent mode

When CL is specified, an intercept path is created for all SVCs (selected by the SVC parameter) issued from the task specified in the NAME or TID parameter.

When RX is specified, an intercept path is created for all SVCs (selected by the SVC parameter) directed to an existing task, device, pseudo task, or pseudo device specified in the NAME parameter.

When RN is specified, a pseudo device is created for SVC 1 or SVC 7, or a pseudo task is created for SVC 6. The pseudo device or task is attached to the intercept path created by the call.

A pseudo task or pseudo device is deleted when all intercept paths attached to it are removed. When a pseudo device is assigned without SVC 7 interception, the requested access privileges are ignored and shared read/shared write privileges are granted. If an SVC 1 is attempted to a pseudo device without an interception in effect, an invalid function error (X'C0') is returned.

NAME=	indicates the address of the memory location specifying the name of a device, task, pseudo device, or pseudo task. This location must be fullword boundary aligned and contain eight bytes of blanks followed by a standard file descriptor (fd) or task identifier (taskid). An fd must be packed, left-justified, and padded with blanks within the fullword. A taskid must be left-justified and padded with blanks. When RX or RN is specified by the MODE parameter, the standard fd or taskid given with the NAME parameter can include an asterisk or a backward slash to allow generic naming. See Section 4.6.

TABLE 4-3 VALID COMBINATIONS FOR SVC, MODE, AND NAME PARAMETERS

ICREATE PARAMETERS			FUNCTION
SVC=	MODE=	NAME=	
(1)	CL	taskid	Intercepts any SVC 1 issued from the task
	RX	fd	Intercepts any SVC 1 directed to the existing device
	RN	fd	Creates a pseudo device and intercepts any SVC 1 directed to it
(2,7)	CL	taskid	Intercepts any SVC 2 code 7 issued from the task
	RX	--	No function; specifying fd or taskid results in error
	RN	--	Results in error
(3)	CL	taskid	End of task interception; occurs no matter how a task terminates
	RX	--	No function; specifying fd or taskid results in error
	RN	--	Results in error

TABLE 4-3 VALID COMBINATIONS FOR SVC, MODE, AND NAME PARAMETERS (Continued)

ICREATE PARAMETERS			FUNCTION
SVC=	MODE=	NAME=	
(6)	CL	taskid	Intercepts any SVC 6 issued from the task
	RX	taskid	Intercepts any SVC 6 directed to the existing task
	RN	taskid	Creates a pseudo task and intercepts any SVC 6 directed to it
(7)	CL	taskid	Intercepts any SVC 7 issued from the task
	RX	fd	Intercepts any SVC 7 directed to the existing device
	RN	fd	Creates a pseudo device and intercepts any SVC 7 directed to it

TID=	indicates the address of a fullword location containing a task identifier. This parameter, which is mutually exclusive with the NAME= parameter, can be used when MODE=CL, or MODE=RX with SVC 6, to identify the task to be intercepted. The TID can be obtained from the RDB.TID field of an RDB from a previously intercepted SVC call.
CONTROL=	contains a mnemonic indicating either full control (FC) or monitor control (MC) over intercepted SVCs. When CONTROL=FC, an intercepting task can exert full control over an application task's intercepted SVCs. When CONTROL=MC, an intercepting task acts as a monitor only; it has no control over an intercepted SVC.
BUFFERL=	indicates the address of the standard circular list that contains the addresses of available RDB buffers.

The RDB used by the intercepting task to identify an intercepted SVC must not be moved to a new location after the interception takes place. The system ensures that the address of this RDB is the same as the address of the RDB that was passed to the intercepting task when the interception occurred.

HANDLER=	indicates the address of a fullword location containing the name of a queue handler. This name, a maximum of eight characters, is left-justified and padded with blanks. If this parameter is omitted, the default queue handler is invoked.

NOTE

Currently, user defined queue handlers are not supported.

PID=	indicates the address of a halfword location that is used by the system to store the path identifier for the intercept path.
EXEC=	is the address of an SVC intercept executor routine within the intercepting task. This routine will process intercepted SVCs of the type specified with the SVC parameter. During SVC interception, the system removes an RDB specified by the list, fills it with information, and queues a task event trap with the specified executor address to the intercepting task. On entry to an executor routine, general register 0 contains the PID of the intercept path and general register 1 contains the address of the RDB buffer associated with the intercepted SVC. The executor routine executes as task event service routine.
PBSIZE=	specifies the number of bytes in the parameter block for the SVC indicated by the SVC parameter. When this parameter is omitted, the parameter block size defaults to the standard sizes documented for each type of SVC in the OS/32 Supervisor Call (SVC) Reference Manual, except for SVC 2 code 7 interception, which defaults to eight bytes.

	The size of the RDB.PB field in the RDB for this interception path is the value of the PBSIZE parameter (or its default if PBSIZE is not specified).
SVAR=	is the address of a fullword location containing user defined data. This data is passed to the intercept logic. The queue handler named by the HANDLER parameter can later access the data. The SVAR parameter is for user defined purposes when needed by a user defined queue handler. **NOTE** Currently, user defined queue handlers are not supported.
ERROR=	is the address of an error routine within the intercepting task. If a run-time error occurs for this macro, execution branches to this error routine. If this parameter is omitted and a run-time error occurs, execution resumes with the instruction following code built by the macro.
PCB=	is the address of a PCB previously constructed and initialized by the FORM=L parameter. When no PCB parameter is included, macro code automatically builds a new PCB and initializes it with values corresponding to the other specified parameters.
FORM=	L requests a PCB to be built but not executed. Macro code constructs a PCB for this macro and initializes it with values. Subsequent macros can reference this PCB via the PCB parameter.
NINTC=	n specifies the number of interceptions that can be handled concurrently for this intercept path. If there are more SVC interceptions outstanding than can be handled concurrently, the excess interceptions are queued. The default value for n is 1.

4.13.2 IREMOVE Macro

The IREMOVE macro allows an intercepting task to remove one or all previously created SVC intercept paths.

Format:

NAME	OPERATION	OPERAND
symbol	IREMOVE	PID=pointer
		,TERM= { CS / AB }
		[,ERROR=pointer]
		[,PCB=pointer]
		[,FORM=L]

Parameters:

PID=	is the address of the path identifier specifying the path being removed. A zero value in the PID halfword removes all existing intercept paths.
TERM=	indicates either of two termination modes for intercepted SVCs already queued for the intercepting task:

- AB indicates abort. OS/32 aborts all currently queued requests before path removal.
- CS indicates controlled shutdown. OS/32 services only currently queued requests before path removal; no requests made after TERM=CS is issued can be queued or processed.

  If this parameter is omitted, AB is the default.

ERROR=	is the address of an error routine within the intercepting task. If a run-time error occurs for this macro, execution branches to this error routine. If this parameter is omitted and a run-time error occurs, execution resumes with the instruction following the macro.

PCB=	is the address of a PCB previously constructed and initialized by the FORM=L parameter. If this parameter is omitted, a new PCB is automatically built and initialized with values corresponding to the other specified parameters.
FORM=	L requests a PCB be built but not executed. A PCB is built by this macro and initialized with values. Subsequent macros can reference this PCB via the PCB parameter.

4.13.3 IGET Macro

The IGET macro allows an intercepting task to get data from the application task whose SVC is intercepted.

Format:

NAME	OPERATION	OPERAND
symbol	IGET	RDB=pointer
		,ADST=pointer
		,ADEND=pointer
		,SDST=pointer
		,SDEND=pointer
		[,ERROR=pointer]
		[,PCB=pointer]
		[,FORM=L]
		[,DONE=addr]

Parameters:

RDB=	is the address of the RDB buffer built for the intercepted SVC.
ADST=	is the start address of a data area within the application task whose SVC is intercepted. The contents of this area are transferred to an intercepting task data area.

ADEND=	is the end address of the data area within the application task whose SVC is intercepted.
SDST=	is the start address of a data area within the intercepting task. This area receives the data from the application task.
SDEND=	is the end address of the data area within the intercepting task.
ERROR=	is the address of an error routine within the intercepting task. If a run-time error occurs for this macro, execution branches to this error routine. If this parameter is omitted and a run-time error occurs, execution resumes with the instruction following the macro.
PCB=	is the address of a PCB previously constructed and initialized by the FORM=L parameter. If this parameter is omitted, a new PCB is automatically built and initialized with values corresponding to the other specified parameters.
FORM=	L requests a PCB be built but not executed. A PCB is built for this macro and initialized with values. Subsequent macros can reference this PCB via the PCB parameter.
DONE=	is an address that specifies that the macro is to be a PROCEED call. When the call is completed, a task event interrupt occurs, using the routine specified by the address in the DONE parameter. This routine enters with R0 containing the error code for the call and R1 pointing to the macro's parameter block. Once this routine has finished processing, it exits using the code built by the TEXIT macro. The proceed form of the IGET macro must be used if an IROLL macro was issued to the application task whose SVC is intercepted. The system cannot guarantee that the application task is in memory or that it can be rolled into memory within a reasonable time.

4.13.4 IPUT Macro

The IPUT macro lets an intercepting task put data into a data area of the application task whose SVC is intercepted.

Format:

NAME	OPERATION	OPERAND
symbol	IPUT	RDB=pointer
		,ADST=pointer
		,ADEND=pointer
		,SDST=pointer
		,SDEND=pointer
		[,ERROR=pointer]
		[,PCB=pointer]
		[,FORM=L]
		[,DONE=addr]

Parameters:

RDB=	is the address of the RDB buffer built for the intercepted SVC.
ADST=	is the start address of a data area within the application task. This area receives the contents of an intercepting task data area.
ADEND=	is the end address of the data area within the application task.
SDST=	is the start address of a data area within the intercepting task. The contents of this area are put into the application task data area.
SDEND=	is the end address of the data within the application task.
ERROR=	is the address of an error routine within the intercepting task. If a run-time error occurs for code built by this macro, execution branches to this error routine.

If this parameter is omitted and a run-time error occurs, execution resumes with the instruction following the macro.

PCB=	is the address of a PCB previously constructed and initialized by the FORM=L parameter. If this parameter is omitted, a new PCB is automatically built and initialized with values corresponding to the other specified parameters.
FORM=	L requests a PCB be built but not executed. A PCB is built for this macro and initialized with values. Subsequent macros can refer to this PCB via the PCB parameter.
DONE=	is an address that specifies that the macro is to be a proceed call. When the call is completed, a task event interrupt occurs, using the routine specified by the address in the DONE parameter. This routine enters with general register 0 containing the error code for the call, and general register 1 pointing to the macro's parameter block. Once this routine has finished processing, it exits using the code built by the TEXIT macro. The proceed form of the IPUT macro must be used if an IROLL macro was issued to the application task. The system cannot guarantee that the application task is in memory or that it can be rolled into memory within a reasonable time.

4.13.5 ICONT Macro

The ICONT macro relinquishes control of an intercepted SVC by returning control to an OS/32 SVC executor.

Format:

NAME	OPERATION	OPERAND
symbol	ICONT	RDB=pointer
		[,ERROR=pointer]
		[,PCB=pointer]
		[,FORM=L]

Parameters:

RDB=	is the address of the RDB buffer built for the intercepted SVC.
ERROR=	is the address of an error routine within the intercepting task. If a run-time error occurs for code built by this macro, execution branches to this error routine. If this parameter is omitted and a run-time error occurs, execution resumes with the instruction following the code built by the macro.
PCB=	is the address of a PCB previously constructed and initialized by the FORM=L parameter. If this parameter is omitted, a new PCB is automatically built and initialized with values corresponding to the other specified parameters.
FORM=	L requests a PCB be built but not accessed. A PCB is built for this macro and initialized with values. Subsequent macros can refer to this PCB via the PCB parameter.

4.13.6 IPROCEED Macro

After an SVC has been intercepted, the intercepting task can execute code built by an IPROCEED macro to allow the application task that issued the SVC to proceed with its execution. Until the intercepting task executes code built by an IPROCEED macro, the application task is in a wait state.

Format:

NAME	OPERATION	OPERAND
symbol	IPROCEED	RDB=pointer
		[,ERROR=pointer]
		[,PCB=pointer]
		[,FORM=L]
		[,CC=n]

Parameters:

RDB=	is the address of the RDB buffer built for the intercepted SVC.
ERROR=	is the address of an error routine within the intercepting task. If a run-time error occurs for code built by this macro, execution branches to this error routine. If this parameter is omitted and a run-time error occurs, execution resumes with the instruction following code built by the macro.
PCB=	is the address of a PCB previously constructed and initialized by the FORM=L parameter. If this parameter is omitted, a new PCB is automatically built and initialized with values corresponding to the other specified parameters.
FORM=	L requests a PCB be built but not assessed. A PCB is built for this macro and initialized with values. Subsequent macros can refer to this PCB via the PCB parameter.
CC=	n is a decimal number specifying the setting of the application task PSW condition code after the SVC instruction execution. If the CC parameter is omitted, the condition code of the application task PSW is set to zero.

4.13.7 IROLL Macro

After an SVC is intercepted, an IROLL macro lets an intercepting task change the status of the application task from nonrollable to rollable, provided that the task was established as rollable by Link. This allows OS/32 to roll out a task having an intercepted SVC that requires lengthy processing.

Format:

```
         |           |
 NAME    | OPERATION |            OPERAND
-------------------------------------------------------
symbol   |   IROLL   | RDB=pointer
         |           |
         |           | [,ERROR=pointer]
         |           |
         |           | [,PCB=pointer]
         |           |
         |           | [,FORM=L]
         |           |
```

Parameters:

RDB=	is the address of the RDB buffer built for the intercepted SVC.
ERROR=	is the address of an error routine within the intercepting task. If a run-time error occurs for this macro, execution branches to this error routine. If this parameter is omitted and a run-time error occurs, execution resumes with the instruction following the macro.
PCB=	is the address of a PCB previously constructed and initialized by the FORM=L parameter. If this parameter is omitted, a new PCB is automatically built and initialized with values corresponding to the other specified parameters.
FORM=	L requests a PCB be built but not accessed. A PCB is built for this macro and initialized with values. Subsequent macros can refer to this PCB via the PCB parameter.

4.13.8 ITERM Macro

The ITERM macro terminates SVC processing. It also allows an intercepting task to return the parameter block of the SVC it processed to the application task that issued the SVC. The returned parameter block can have updated information such as status, number of bytes transferred, etc.

Format:

NAME	OPERATION	OPERAND
symbol	ITERM	RDB=pointer
		,TRAP=pointer
		$,COPY=\left\{\begin{matrix} Y \\ N \end{matrix}\right\}$
		[,ERROR=pointer]
		[,PCB=pointer]
		[,FORM=L]
		[,CC=n]

Parameters:

RDB=	is the address of the RDB buffer built for the intercepted SVC.
TRAP=	is the address of a fullword that contains an item to be added to the task queue of the application task whose SVC is intercepted.
COPY=	Y (yes) indicates that the SVC parameter block in the RDB is to be copied back into the parameter block of the intercepted SVC. N (no) indicates the copy operation is not performed. If this parameter is omitted, N is the default.
ERROR=	is the address of an error routine within the intercepting task. If a run-time error occurs for code built by this macro, execution branches to this error routine. If this parameter is omitted and a run-time error occurs, execution resumes with the instruction following the code built by the macro.
PCB=	is the address of a PCB previously constructed and initialized by the FORM=L parameter. If this parameter is omitted, a new PCB is automatically built and initialized with values corresponding to the other specified parameters.
FORM=	L requests a PCB be built but not accessed. A PCB is built for this macro and initialized with values. Subsequent macros can refer to this PCB via the PCB parameter.
CC=	n is a decimal number specifying the setting of the application task PSW condition code after the SVC instruction execution. If the CC parameter is omitted, the condition code of the application task PSW is set to zero.

4.13.9 ITRAP Macro

The ITRAP macro allows an intercepting task to send a task queue item to an application task whose SVC is intercepted. The task queue item can be any of the task queue items supported by OS/32.

Format:

```
          |              |
  NAME    |  OPERATION   |            OPERAND
--------------------------------------------------------
 symbol   |    ITRAP     | (RDB=pointer)
          |              | {           }
          |              | (TID=pointer)
          |              |
          |              |  ,TRAP=pointer
          |              |
          |              | [,ERROR=pointer]
          |              |
          |              | [,PCB=pointer]
          |              |
          |              | [,FORM=L]
          |              |
          |              | [,DONE=addr]
          |              |
```

Parameters:

RDB=	is the address of the RDB buffer built for the intercepted SVC.
TID=	is the address of a fullword containing the taskid for the task. Before issuing an ITRAP macro with the TID parameter, the intercepting task must have obtained the task identifier from an RDB and placed it into the fullword location.

NOTE

The TID form of this macro can be used to send a trap to a task that is not being intercepted.

TRAP=	is the address of a fullword that contains an item to be added to the task queue of the application task having an SVC that is intercepted.
ERROR=	is the address of an error routine within the intercepting task. If a run-time error occurs for code built by this macro, execution branches to this error routine. If this parameter is omitted and a run-time error occurs, execution resumes with the instruction following the code built by the macro.
PCB=	is the address of a PCB previously constructed and initialized by the FORM=L parameter. If this parameter is omitted, a new PCB is automatically built and initialized with values corresponding to the other specified parameters.
FORM=	L requests a PCB be built but not accessed. A PCB is built for this macro and initialized with values. Subsequent macros can refer to this PCB via the PCB parameter.
DONE=	is an address that specifies that the macro is to be a PROCEED call. When the call is completed, a task event interrupt occurs, using the routine whose address is specified in the DONE parameter. This routine enters with general register 0 containing the error code for the call and general register 1 pointing to the macro's parameter block. Once this routine has finished processing, the intercepting task exits using code built by the TEXIT macro. The proceed form of the ITRAP macro must be used if an IROLL macro was specified in the application task having an SVC that is intercepted. The system cannot guarantee that the application task is in memory or that it can be rolled into memory within a reasonable time.

4.13.10 IERRTST Macro

The IERRTST macro allows an intercepting task to evaluate errors
| resulting from execution of code built by intercept macros in order to branch to appropriate error handling routines.

Format:

```
          |             |
   NAME   |  OPERATION  |            OPERAND
----------------------------------------------------
 symbol   |   IERRTST   | xx=pointer
          |             |       .
          |             |       .
          |             |       .
          |             | [xx=pointer]
          |             |
          |             | [ELSE=pointer]
          |             |
          |             | [PCB=pointer]
          |             |
          |             | [FORM=L]
          |             |
```

Parameters:

xx=	is a two-character alphabetic string specifying one of the error codes for the intercept macros. See Table 4-2.
Pointer	specifies the name of an intercepting task error routine that handles errors having a returned error code identical to the one specified by the xx parameter. For instance, an IERRTST macro might include these parameters for evaluating an IPUT macro:

```
IERRTST    AD=pointer,NT=pointer,RD=pointer
```

These parameters specify the addresses of the error routines to which execution will branch whenever the returned error code equals AD, NT, or RD.

ELSE=	is the name of an error routine to be executed for errors other than those specified in the xx parameter. If this parameter is omitted, either of the following actions occurs for returned errors: • If the returned error code corresponds to the one specified by the xx parameter, execution branches to a specific error routine. • If the returned error code does not correspond to the one specified by the xx parameter, execution branches to the instruction immediately following the code built by the IERRTST macro.
PCB=	is the address of a PCB previously constructed and initialized by the FORM=L parameter. If this parameter is omitted, a new PCB is automatically built and initialized with values corresponding to the other specified parameters.
FORM=	L requests a PCB be built but not accessed. A PCB is built for this macro and initialized with values. Subsequent macros can refer to this PCB via the PCB parameter.

4.13.11 $RDB Macro

The $RDB macro is used to define a structure containing the symbolic names for all of the RDB fields. It is recommended that symbolic names be used to refer to the RDB fields instead of coding the hexadecimal offsets to the fields.

Format:

NAME	OPERATION	OPERAND
symbol	$RDB	

4.14 SAMPLE SUPERVISOR CALL (SVC) INTERCEPTION PROGRAMS

The following program uses SVC interception software to intercept SVC 1 to the existing real device MAG1. Each time an SVC 1 is issued to MAG1, the program prints out the following message:

```
SVC 1 CALL INTERCEPTED
```

The SVC 1 is terminated with a device unavailable error code (X'AO').

```
        $RDB                       DEFINES AN RDB STRUCTURE

*  ADD AN RDB BUFFER ADDRESS TO THE RDB BUFFER ADDRESS LIST.

        LA      O,RDB              LOAD THE ADDRESS OF THE RDB
*                                  INTO REGISTER 0

        ABL     O,BUFLIST          ADD THE ADDRESS OF THE RDB
*                                  TO THE CIRCULAR LIST

*  CREATE THE INTERCEPT PATH

        ICREATE NAME=INTNAME,      FD FOR DEVICE NAME                   X

                MODE=RX,           RECIPIENT-EXISTENT MODE              X

                CONTROL=FC,        GIVES INTERCEPTING TASK FULL CONTROL X

                SVC=(1),           ALL SVC 1 ARE TO BE INTERCEPTED      X

                EXEC=INTRTN,       POINTS TO THE SVC EXECUTOR ROUTINE   X

                BUFFERL=BUFLIST,   ASSIGNS POINTER TO FREE BUFFER LIST  X

                PID=PATHID,        DATA AREA FOR INTERCEPT PATH ID      X

                ERROR=BOMBOUT      ERROR ROUTINE FOR ICREATE MACRO

*  IF ERROR OCCURS IN ICREATE MACRO ENABLE TASK EVENT TRAP SO  TASK
*  CAN GO INTO TRAP WAIT FOR INTERCEPTS TO OCCUR

*  LOAD TSW WITH WAIT STATE SET AND TASK EVENT TRAPS ENABLED

        LTSW TETS,WT

*  COME HERE IF ERROR OCCURS IN ICREATE MACRO
```

```
BOMBOUT  SVC 3,1                     FAIL TASK ON ERROR

*  ALLOCATE DATA AREA FOR ICREATE

         ALIGN 4
INTNAME  DC     C'    '              NODE NAME
         DC     C'    '              RESERVED
         DC     C'MAG1'              DEVICE NAME
         DC     C'    '              FILE NAME PART 1
         DC     C'    '              FILE NAME PART 2
         DC     C'    '              EXTENSION

BUFLIST DLIST 1                      DESIGNATE 1 RDB IN CIRCULAR LIST

RDB    DS RDB.PB+20                  ALLOCATES SIZE OF RDB + SVC 1

PATHID  DS 2                         DESIGNATE AREA FOR PATH ID

*  TRAP EVENT SERVICE ROUTINE
*  THE FOLLOWING ROUTINE IS EXECUTED WHEN AN SVC IS INTERCEPTED

INTRTN     SVC 2,NOTIFY              LOG MESSAGE THAT SVC 1 WAS INTER-
*                                    CEPTED
           LHI  0,X'A000'            RETURN DEVICE UNAVAILABLE STATUS
*                                    FOR INTERCEPTED SVC 1
           STH  0,RDB.PB+2(1)        SAVE SVC 1 STATUS IN STATUS FIELD
*                                    OF RDB

*
* TERMINATE THE INTERCEPTED CALL, COPYING THE MODIFIED SVC
* PARAMETER BLOCK IN THE RDB BACK OVER THE USER'S SVC PARAMETER
* BLOCK.

         ITERM RDB=(1),COPY=Y

         TEXIT                       EXIT THE TASK EVENT ROUTINE

*  ALLOCATE DATA AREA FOR TRAP EVENT SERVICE ROUTINE
           ALIGN 4
NOTIFY     DB  0,7,0,22
           DC   C'SVC 1 CALL INTERCEPTED'
           END
```

The following program creates a pseudo device to which a u-task can assign and write. The user's data buffer is passed to the OS/32 command processor via SVC 2 code 14 to be executed as a command line.

```
IRDR      PROG  SVC INTERCEPT EXAMPLE - INTERNAL READER
**********************************************************************
**********************************************************************
*                                                                    *
*                                                                    *
*         This task creates a pseudo device to which a user task     *
*         can assign and write.  The user's data buffer is           *
*         passed to the OS command processor via a SVC 2,14 to       *
*         be executed as a command line.                             *
*                                                                    *
*                                                                    *
**********************************************************************
**********************************************************************
R00       EQU    0
R01       EQU    1
R02       EQU    2
R03       EQU    3
R04       EQU    4
R05       EQU    5
R06       EQU    6
R07       EQU    7
R08       EQU    8
R09       EQU    9
R10       EQU    10
R11       EQU    11
R12       EQU    12
R13       EQU    13
R14       EQU    14
R15       EQU    15
          SPACE  3
          NLSTM
          NLSTU
          $SVC1
          $SVC7
          $RDB
```

```
        TITLE INTERCEPT PATH CREATION
**********************************************************************
*                                                                    *
*       SET UP INTERCEPT PATHS                                       *
*                                                                    *
**********************************************************************
IRDR    EQU   *
        SVC   2,PEEK01             GET NAME OF SYSTEM CONSOLE
        L     R00,CON
        ST    R00,SVC7.VOL+SVC7CON
        LHI   R00,SV7.ASGN!SV7.SRW
        SLL   R00,16               ASSIGN LU 0 SRW
        ST    R00,SVC7.OPT+SVC7CON
        SVC   7,SVC7CON            ASSIGN TO SYSTEM CONSOLE
        LB    R00,SVC7.STA+SVC7CON
        LR    R00,R00              WAS THE ASSIGN OK?
        BNZ   BADCON               NO
        LIS   R00,0                CHANGE SVC 7 TO FETCH ATTR
        STH   R00,SVC7.OPT+SVC7CON
        SVC   7,SVC7CON            FETCH ATTRIBUTES ON CON:
        LB    R00,SVC7.STA+SVC7CON
        LR    R00,R00              WAS THE FETCH OK?
        BNZ   BADCON               NO
        LHI   R00,SV7.CLOS         CHANGE SVC 7 TO CLOSE
        SRLS  R00,8                DO NOT DESTROY DEVICE CODE
        STB   R00,SVC7.OPT+SVC7CON
        SVC   7,SVC7CON            CLOSE THE SYSTEM CONSOLE
        LB    R00,SVC7.STA+SVC7CON
        LR    R00,R00              WAS THE CLOSE OK?
        BNZ   BADCON               NO
        LHI   R00,X'7FFF'          BAD LENGTH FOR SVC 2,14 TO GET
        STH   R00,COMMAND+4        MAX LENGTH ALLOWED BY SYSTEM
        SVC   2,COMMAND            WILL GET ERROR STATUS 3
        LH    R00,COMMAND+6        USE AS IRDR LENGTH
        STH   R00,SVC7.LRC+SVC7CON
        SPACE 1
        LHI   R00,RDBNUM           NUMBER OF RDB'S
        LA    R01,RDBPOOL          ADDRESS OF RDB POOL
INTRDB  EQU   *
        ATL   R01,RDBP             ADD RDB TO QUEUE
        AHI   R01,RDBSIZE          ADDRESS OF NEXT RDB
        SIS   R00,1                ALL RDB'S ADDED TO QUEUE?
        BNZ   INTRDB               NO
        SPACE 1
        ICREATE SVC=(7),MODE=RN,NAME=NAME,                                X
              CONTROL=FC,BUFFERL=RDBP,PID=PID,EXEC=INT7
        IERRTST FD=BADFD,EX=BADEX,ELSE=BADALL
        ICREATE SVC=(1),MODE=RX,NAME=NAME,PBSIZE=SVC1X,                   X
              CONTROL=FC,BUFFERL=RDBP,PID=PID,EXEC=INT1
        IERRTST FD=BADFD,EX=BADEX,ELSE=BADALL
        SPACE 1
        LTSW  WT,TETS              ENTER TRAP WAIT
        SPACE 3
```

```
BADFD    SVC   2,LOGFD
         SVC   3,1
BADEX    SVC   2,LOGEX
         SVC   3,1
BADALL   SVC   2,STRANGE
         SVC   3,1
BADCON   SVC   2,LOGCON
         SVC   3,1
         SPACE 1
         ALIGN 4
LOGFD    DC    H'7',H'8'
         DC    C'FD ERROR'
LOGEX    DC    H'7',H'8'
         DC    C'EX ERROR'
STRANGE  DC    H'7',H'8'
         DC    C'!! ERROR'
LOGCON   DC    H'7',H'12'
         DC    C'!!CON ERROR '
         SPACE 1
NAME     DC    C'        IRDR               '
PID      DSF   1
         SPACE 1
RDBNUM   EQU   3                  NUMBER OF RDB'S IN POOL
RDBSIZE  EQU   RDB.+SVC7.         MAXIMUM SIZE OF RDB
RDBP     DLIST RDBNUM             RDB POOL
RDBPOOL  DS    RDBSIZE*RDBNUM     RDB BUFFERS
```

```
        TITLE SVC 7 TEQ HANDLER
*********************************************************************
*                                                                   *
*       SVC 7 INTERCEPT EXECUTOR                                    *
*                                                                   *
*********************************************************************
INT7    EQU   *
        LR    R10,R01                SAVE RDB POINTER
        LR    R11,R10
        AH    R11,RDB.OFF(R10)       ADDRESS OF SVC 7 PBLK
        LB    R00,SVC7.OPT(R11)      GET SVC 7 OPTIONS
        LR    R00,R00                FETCH ATTRIBUTES?
        BZ    DOFETCH                YES
        CLHI  R00,X'FF'              EXTENDED SVC 7 FUNCTIONS?
        BE    INT7.NS                YES - NOT SUPPORTED
        THI   R00,X'40'              ASSIGN?
        BNZ   DOOPEN                 YES
        THI   R00,X'04'              CLOSE?
        BNZ   DOCLOSE                YES
        THI   R00,X'21'              CHAP OR CHECKPOINT?
        BNZ   INT7.IG                YES - IGNORE
        SPACE 1
INT7.NS EQU   *
        SVC   2,UNPACK7              PUT SVC 7 OPTION IN ERROR MESSAGE
        SVC   2,LOG7ERRC             AND LOG ERROR MESSAGE
        LIS   R00,1                  RETURN ILLEGAL FUNCTION TO USER
        STB   R00,SVC7.STA(R11)      AS AN ERROR STATUS
        ITERM PCB=TERM,RDB=(R10)     TERMINATE THIS SVC 7
        TEXIT PCB=EXIT               EXIT FROM TEQ HANDLER
        SPACE 3
*
*       IGNORE SVC 7 COMMAND PROCESSOR
*
INT7.IG EQU   *
        ITERM PCB=TERM,RDB=(R10)     IGNORE THIS SVC 7
        TEXIT PCB=EXIT               EXIT FROM TEQ HANDLER
        SPACE 3
```

```
*
*        OPEN PROCESSOR
*
DOOPEN   EQU   *
         LB    R15,SVC7.OPT+1(R11) GET ACCESS PRIVILEGES
         SRLS  R15,5               SRO = 0   &   ERO = 1
         CLHI  R15,2               REQUESTING READ ONLY ACCESS?
         BL    OPEN.ERR            YES - ERROR
         B     OPEN.OK             SKIP SECURITY CHECK
         SPACE 2
*--------------USER DEFINED SECURITY CHECK FOLLOWS-------------------
         L     R15,RDB.TID(R10)    MOVE TID FOR PEEK03
         ST    R15,TID
         SVC   2,PEEK03            INFO ON USER TASK
         LM    R14,MONITOR         GET NAME OF USERS MONITOR
         CLI   R14,C'.MTM'         TASK A SUB-TASK OF MTM?
         BNE   OPEN.OK             NO
         CLI   R15,C'    '         BE SURE
         BNE   OPEN.OK             NO?
         LM    R14,TASKNAME        GET NAME OF USER
         CLI   R14,C'LEE '         IS IT ME?
         BNE   OPEN.ERR            NO
         CLI   R15,C'    '         BE SURE
         BNE   OPEN.ERR            NO?
         L     R15,LEGACY          GET NAME OF USERS TERMINAL
         CLI   R15,C'CT42'         IS IT MINE?
         BNE   OPEN.ERR            NO
         L     R15,ACCT.P          GET USERS PRIVATE ACCOUNT NUMBER
         CLHI  R15,29              AM I IN MY ACCOUNT?
         BNE   OPEN.ERR            NO
         L     R15,ACCT.G          GET USERS GROUP ACCOUNT NUMBER
         CLHI  R15,18              DO I HAVE MY CORRECT GROUP ACCOUNT?
         BNE   OPEN.ERR            NO
*------------------------------------------------------------------
         SPACE 2
OPEN.OK  EQU   *
         ICONT PCB=CONT,RDB=(R10)  RETURN TO OS SVC 7 EXECUTOR
         TEXIT PCB=EXIT            EXIT FROM TEQ HANDLER
         SPACE 2
OPEN.ERR EQU   *
         LIS   R15,9               RETURN ASSIGNMENT ERROR TO USER
         STB   R15,SVC7.STA(R11)
         ITERM PCB=TERM,RDB=(R10)  RETURN BAD STATUS TO USER
         TEXIT PCB=EXIT            EXIT FROM TEQ HANDLER
         SPACE 3
```

```
*
*         CLOSE PROCESSOR
*
DOCLOSE   EQU    *
          ICONT PCB=CONT,RDB=(R10)  RETURN OS OS SVC 7 EXECUTOR
          TEXIT PCB=EXIT            EXIT FROM TEQ HANDLER
          SPACE 3
*
*         FETCH ATTRIBUTES PROCESSOR
*
DOFETCH   EQU    *
          LA     R09,SVC7CON        GET ADDRESS OF FETCH ATTR OF CON
          LB     R15,SVC7.OPT+1(R09) MOVE DEVICE CODE
          STB    R15,SVC7.OPT+1(R11)
          LIS    R15,0              GOOD STATUS
          STB    R15,SVC7.STA(R11)
          L      R15,SVC7.KEY(R09)  DEVICE ATTR & RECORD LENGTH
          ST     R15,SVC7.KEY(R11)
          L      R15,NAME+8         IRDR DEVICE NAME
          ST     R15,SVC7.VOL(R11)
          LM     R12,SVC7.FNM(R09)
          STM    R12,SVC7.FNM(R11)
          ITERM PCB=TERM,RDB=(R10)  RETURN SVC 7 FETCH PBLK TO USER
          TEXIT PCB=EXIT            EXIT FROM TEQ HANDLER
```

```
        TITLE SVC 1 TEQ HANDLER
**********************************************************************
*                                                                    *
*       SVC 1 INTERCEPT EXECUTOR                                     *
*                                                                    *
**********************************************************************
INT1     EQU   *
         LR    R10,R01              SAVE RDB ADDRESS
         LR    R07,R10
         AH    R07,RDB.OFF(R10)     ADDRESS OF SVC 1 PBLK
         LIS   R14,0                NO ERROR ON COMMAND FUNCTION
         LIS   R15,0                LENGTH OF TRANSFER
         LB    R13,SVC1.FC(R07)     GET FUNCTION CODE
         THI   R13,SV1.CMDF         COMMAND FUNCTION?
         BNZ   ECHODONE             YES - TREAT AS A NOP
         THI   R13,SV1.WRIT         IS USER DOING A WRITE?
         BNZ   INT1.WRT             YES
*
*        A read from the internal reader will give the
*        user an illegal function status.
*
         LHI   R14,X'C000'          ILLEGAL FUNCTION ON READ
         B     ECHODONE             FINISH UP
*
*        Queue the user's command line to the internal reader
*
INT1.WRT EQU   *
         L     R11,SVC1.SAD(R07)    GET START ADDRESS
         L     R12,SVC1.EAD(R07)    AND END ADDRESS
         IGET  RDB=(R10),SDST=BUFFER,SDEND=BUFEND,                     *
               ADST=(R11),ADEND=(R12)
         SR    R12,R11              GET LENGTH-1 OF STRING
         LR    R15,R12
         AIS   R15,1                LENGTH OF USER COMMAND LINE
         STH   R15,COMMAND+4
         SVC   2,COMMAND            PASS COMMAND TO IREADER
         LH    R14,COMMAND+2        COMMAND QUEUED TO IREADER?
         BZ    ECHODONE             YES
         LHI   R14,X'A000'          NO  - GIVE DEVICE UNAVAILABLE
         SPACE 1
ECHODONE EQU   *
         STH   R14,SVC1.STA(R07)    RETURN STATUS
         ST    R15,SVC1.LXF(R07)    RETURN LENGTH
         THI   R13,SV1.WAIT         IS USER REQUEST A WAIT?
         BNZ   ECHOWAIT             YES - NO NEED FOR A TRAP
         SPACE 1
         L     R15,RDB.PAD(R10)     GET ADDRESS OF USER SVC 1 PBLK
         OI    R15,Y'08000000'      I/O PROCEED COMPLETION PARAMETER
         ST    R15,TRAP
         SPACE 1
         ITERM RDB=(R10),TRAP=TRAP,COPY=Y TERMINATE WITH TRAP
         TEXIT PCB=EXIT             EXIT FROM TEQ HANDLER
         SPACE 1
ECHOWAIT EQU   *
         ITERM PCB=TERM,RDB=(R10)   TERMINATE THIS SVC 1
         TEXIT PCB=EXIT             EXIT FROM TEQ HANDLER
         EJECT
         ALIGN 4
UNPACK7  DB    2,6,0,0                PUT SVC 7 ERROR CODE IN
         DAC   SVC7ERRC
LOG7ERRC DB    0,7
         DC    Z(LOG7ERRX-*)
         DB    C'UNSUPPORTED SVC 7 FUNCTION '
SVC7ERRC DB    C'.. INTERCEPTED '
LOG7ERRX EQU   *-1
         SPACE 2
         ALIGN 4
TRAP     DS    4                      I/O PROCEED COMPLETION TRAP
         SPACE 2
         ALIGN 4
CONT     ICONT FORM=L                 CONTINUE SVC
         SPACE 2
         ALIGN 4
TERM     ITERM FORM=L,COPY=Y          TERMINATE SVC
         SPACE 2
         ALIGN 4
EXIT     TEXIT FORM=L                 EXIT SVC
         SPACE 2
         ALIGN 4
PEEK01   DB    1,19
         DS    22
CON      DS    4                      SYSTEM CONSOLE NAME
         SPACE 2
         ALIGN 4
SVC7CON  DS    SVC7.
         SPACE 2
         ALIGN 4
PEEK03   DB    3,19,0,0               GET INFO AN USER TASK
TID      DSF   1                      USER TASK ID
TASKNAME DSF   2                      NAME OF USER TASK
CTSW     DSF   1                      CURRENT TASK STATUS WORD
TOPT     DSF   1                      TASK OPTIONS
WAITS    DSF   1                      TASK WAITS
ACCT.P   DSF   1                      USER'S PRIVATE ACCOUNT NUMBER
ACCT.G   DSF   1                      USER'S GROUP ACCOUNT NUMBER
L.VOL    DSF   1                      LOAD VOLUME NAME
L.FD     DSF   2                      LOAD FILE NAME
L.EXT    DSF   1                      LOAD EXTENSION & FILE CLASS
MONITOR  DSF   2                      NAME OF MONITOR TASK
LEGACY   DSF   1                      NAME MTM USERS TERMINAL
PRIO     DS    1                      TASK PRIORITY
         DS    3                      (RESERVED)
         SPACE 3
         ALIGN 4
COMMAND  DB    1,14,0,0               QUEUE COMMAND TO IREADER
         DCX   0                      STATUS
         DCX   0
         DC    A(BUFFER)              ADDRESS OF BUFFER
         SPACE 1
         ALIGN 4
BUFFER   DS    128
BUFEND   EQU   *-1
         END   IRDR
```

CHAPTER 5
OS/32 SUPPORTED INPUT/OUTPUT (I/O) DEVICES

5.1 INTRODUCTION

This chapter discusses the functional aspects of the devices supported by OS/32. Specific device dependent information is included.

OS/32 devices and files support ASCII formatting, sequential access, unconditional and conditional proceed I/O, and vertical forms control (VFC). Device codes associated with Perkin-Elmer supported devices range from 0 through 255. These codes are defined in the System Generation/32 (SYSGEN/32) Reference Manual.

5.2 VERTICAL FORMS CONTROL (VFC)

VFC provides a means to control the vertical forms motion on an output device, such as a line printer or CRT, while writing data. Available VFC functions are:

- Set vertical tabs (EVFU)
- Vertical space 0-79 before or after printing
- Vertical tab before or after printing
- No space before or after printing (overprint)
- Select VFU channels 2-12 before and after printing
- Horizontal tabs (available with BIOC and local line printer drivers only)

The VFC character is the first character of the user's output buffer and is interpreted to mean one of the above functions. VFC characters supported by OS/32 drivers are listed in Appendix B. Other bytes in the buffer are considered to be data and are output without further interpretation.

The OS/32 routines that control VFC can be shared by all drivers requiring VFC character recognition. OS/32 makes no assumption as to the type of device calling the routines; device specificity is maintained by each individual driver.

5.2.1 Horizontal Tabs

When the BIOC driver encounters a horizontal tab character, the driver replaces the character with one or more spaces, as determined by the tab stops that were established by the last down line load. If a horizontal tab character is encountered at a column position beyond the last tab stop, it will be replaced by one space. If a horizontal tab character is encountered while positioned on a tab stop, the necessary number of spaces will be output to position to the next tab stop.

The local line printer driver expands the tab character (control I) to the appropriate number of spaces. Tab stops are defined to be every eighth column; i.e., columns 9, 17, 25, etc. This feature is enabled via extended device code X1 for device codes 112, 113, or 114 only. All other drivers output the horizontal tab character unmodified.

5.2.2 Theory of Operation

For devices that support ASCII output operations (e.g., line printer), a write operation begins with a call to the write initialization routine to determine if there is any VFC operation to be performed before printing.

If a VFC character is present, the driver performs the VFC operation designated by that character. The driver then outputs the user's data buffer. On completion, the driver checks for any VFC operations that are to be performed after the data is output. If a VFC operation is required, the driver performs it. If no "after" VFC operation is required, but the current output is VFC, the driver enters into a line feed pending state for the next write operation.

For drivers that support both input and output operations (e.g., CRT driver), output operations are performed in the same manner as above. However, the procedure for input operations differs slightly. Before an input operation, the cursor remains positioned where the last output operation left it. To prevent the characters that are input from overwriting the previous line, the drive delays echoing the first character input until a line feed is output. Two types of echoing can be performed:

- Software-echo (e.g., BIOC drivers)
- Hardware-echo (e.g., ITAM PPSM drivers)

If a driver uses the software-echo feature, (i.e., the driver echoes the characters that are typed in via software control), the driver waits for the first character to be typed in by the operator. After the character is typed in, the driver performs a VFC operation if it is in line feed pending state before the character is echoed.

If a driver uses the hardware-echo feature and is in line feed pending state, the driver first turns off the echo, waits for the input character to be typed, performs the VFC operation, outputs the character just typed in, and finally turns the hardware-echo back on for the remainder of the buffer.

5.3 CARD READERS

Perkin-Elmer card readers can accommodate a fixed record length of 80 bytes (ASCII), 120 bytes (binary), or 160 bytes (image).

During read ASCII operations, each card column (12 bits) is converted into one 8-bit ASCII character. Illegal codes are converted into the null character (X'00') indicating an error has occurred.

During read binary operations, each pair of card columns (12 bits each) is unpacked into three bytes having the following format:

Read Binary Format:

```
                   First card column
  ------------------------------------------------
 | 12| 11| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
  ------------------------------------------------
 Bytes:
  0                            1

                   Second card column
  ------------------------------------------------
   12| 11| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
  ------------------------------------------------
 Bytes:
                  2
```

During read image operations, each column is converted into one halfword in the following format (U=undefined):

Read Image Format:

```
  ---------------------------------------------------------------
 | U | U | 12| 11| 0 | 1 | 2 | 3 | U | U | 4 | 5 | 6 | 7 | 8 | 9 |
  ---------------------------------------------------------------
 Bits:
   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
```

The translation for an ASCII read is accomplished through a translation table. Devices without hardware translation translate 029- or 026-compatible Hollerith code to 8-bit ASCII code. Source sysgen options include translation of 029- or 026-compatible Hollerith code to EBCDIC code. The hardware translation matches that of the 029-compatible Hollerith to EBCDIC translation.

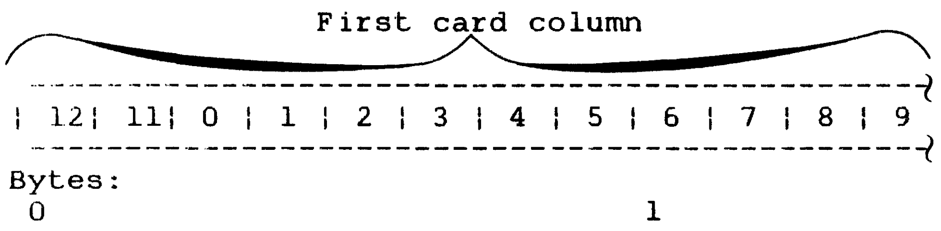
5.4 CARD READER/PUNCH DEVICES

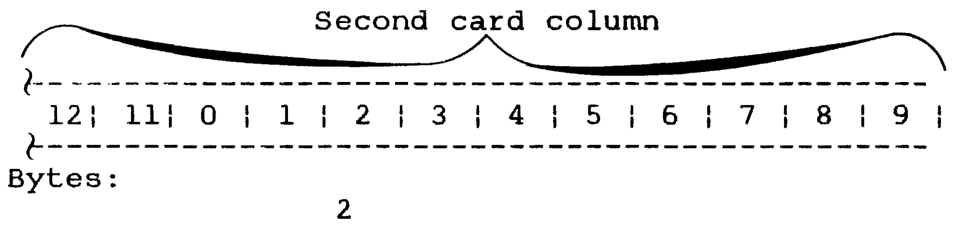
Card reader/punch devices supported by Perkin-Elmer 32-bit computers accommodate fixed record lengths of 80 bytes (ASCII), 120 bytes (column binary), and 160 bytes (image).

During read ASCII operations, each card column (12 bits) is converted into one 8-bit ASCII character. Illegal codes are converted into the null character (X'00') indicating an error has occurred.

During read binary operations, each pair of card columns (12 bits each) is unpacked into three bytes having the following format:

Read Binary Format:





During read image operations, each card column (12 bits each) is placed into a halfword in the following format:

Read Image Format:

```
 ---------------------------------------------------------------
| 0 | 0 | 0 | 0 | 12| 11| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
 ---------------------------------------------------------------
Bits:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
```

During write ASCII operations, each byte of data is translated from ASCII into a 12-bit Hollerith code. Depending on the device code chosen, the following can occur:

- All data is punched and printed.
- Data is punched only.
- Of each 160 bytes of data accepted, the first 80 bytes are punched while the second 80 bytes are printed.

During write binary operations, each 3-byte group is packed into two columns on the card in the following format. Nothing is printed on top of the card.

Write Binary Format:

```
                        Odd Column
 ------------------------------------------------------
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3
 ------------------------------------------------------
Bytes:
 0                                 1

                        Even Column
 ------------------------------------------------------
  4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
 ------------------------------------------------------
Bytes:
                  2
```

During write image operations, the low order 12 bits of each halfword are punched according to the following format. Nothing is printed on top of the card. Bits 0 through 3 are ignored.

Write Image Format:

```
 -----------------------------------------------------------------
|   |   |   |   | 12| 11| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
 -----------------------------------------------------------------
Bits:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
```

The translation for ASCII operations is accomplished through a translation table. The standard translation is 8-bit ASCII code to 029-compatible Hollerith code.

Source sysgen options include 8-bit ASCII code to 026-compatible Hollerith code and also EBCDIC code to 026- or 029-compatible Hollerith codes.

5.5 TELETYPE (TTY) READER/PUNCH

Perkin-Elmer TTY reader/punch devices support read and write ASCII, read and write binary, and read and write image operations. Variable length records are also accommodated.

During read ASCII operations, an X-ON character is output to turn the reader on. The tape is read in blocked mode so data is not copied to the printer while it is being read. Leading blank frames and delete characters are ignored. Data is masked to 7-bit ASCII. The transfer is terminated on buffer full or when a carriage return character is read, whichever occurs first. On termination of the transfer, the tape is advanced to the next delete character or blank frame. An X-OFF character is output to stop the tape.

During read binary operations, an X-ON character is output to turn on the tape. The tape is skipped until the first nonblank frame is found. If the first nonblank character is X'F0', subsequent frames are read in until the user buffer is full. The characters are read in unzoned binary format.

If the first nonblank character read is not X'F0', the characters are read in zoned binary format, stripped of their zones, and packed into the user buffer. Transfer begins with the first nonblank frame after X'F0'. Only punches X'90', X'81' through X'84', and X'95' through X'9F' are read. Other characters are ignored. When the user buffer is full, the tape is advanced to the next blank frame.

During read image operations, none of the above formatting operations are performed. An X-ON character is output to turn the tape on, and data is read into the user buffer until the buffer is full. The X-OFF character is then output to turn the tape off and the transfer is complete.

During write ASCII operations, the driver outputs a RUBOUT-TAPE RUBOUT-RUBOUT sequence in order to initialize the TTY reperforator. Eight frames of blank tape are output as leader, the user data is output until the buffer is empty, or a carriage return character is encountered, whichever occurs first. The driver ensures that a CR-LF-TAPE OFF-RUBOUT sequence terminates the record.

During write binary operations, the driver outputs a RUBOUT-TAPE-RUBOUT-RUBOUT sequence, followed by eight blank frames of leader. The user buffer is output, translating each byte into two frames of zoned binary data. The transfer is terminated when the buffer is empty. The driver outputs a TAPE OFF-RUBOUT sequence.

During write image operations, none of the above formatting or control operations are performed. The user buffer is output until the buffer is empty.

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

Since the reader/punch portion of the TTY is connected to the keyboard/printer portion, only one of these devices can be active at a time. On ASCII write, the data punched on the tape is also printed on the printer.

5.6 TELETYPE (TTY) KEYBOARD/PRINTER

Perkin-Elmer TTY keyboard/printers accommodate variable length records and can be interfaced to current loop devices.

In non-VFC read ASCII operations, data read is masked to 7-bit ASCII. Data is read until the buffer is full or a carriage return is found, whichever occurs first. Upon termination, a carriage return/line feed (CR/LF) sequence is sent to the printer.

In non-VFC write ASCII operations, the buffer is scanned to eliminate trailing blanks. Data is then output until the buffer is exhausted or until a carriage return is found in the data stream. A line feed is automatically appended to the detected carriage return; or if no carriage return is detected, a CR/LF sequence is output after the last nonblank character.

During non-VFC image I/O, none of the above formatting actions occur. The amount of data requested is printed or read in, without masking to 7-bit ASCII, eliminating trailing blanks, checking for control characters, or detecting or appending carriage returns or line feeds. On image read, however, a carriage return is detected as an end of line sentinel.

For information on I/O operations with VFC, see Section 5.2. See the OS/32 Operator Reference Manual for an explanation of the function control keys available on Perkin-Elmer TTY keyboard/printers.

While the reader/punch of an ASR TTY is treated as a separate device, it cannot operate simultaneously with the keyboard/printer.

5.7 PAPER TAPE EQUIPMENT

Variable record lengths are supported by Perkin-Elmer paper tape devices. During read ASCII operations, leading blank tape and delete characters are ignored. Data is masked to 7-bit ASCII. Carriage return terminates read. On termination, the tape is advanced until either a blank frame or a delete character is read.

During read binary operations, tape is advanced until a nonzero character is read. If this character is X'F0', the tape is read until the buffer is full (unzoned binary). If the first nonzero character is not X'F0', the tape is treated as a zoned binary tape. Each two characters are stripped of their zones, merged into one byte, and placed in the buffer until the buffer is full. On buffer full, the tape is advanced until blank tape is found. In zoned binary mode, the only valid characters are: X'90', X'81' - X'84' and X'95' - X'9F'. All other characters cause the transfer to end with a transfer error status.

During read image operations, the tape is read until the buffer is full.

During write ASCII operations, eight frames of blank tape are output. The user buffer is output up to (but not including) carriage return or until the buffer is empty. CR/LF is then output.

During write binary operations, eight frames of blank tape are output followed by the character X'F0'. The user buffer is output until the buffer is empty.

During write image operations, the user buffer is output until the buffer is empty.

5.8 LINE PRINTERS

Perkin-Elmer line printers support variable record lengths up to 132 bytes.

During non-VFC write ASCII operations, the user buffer is output until a carriage return is found or until the buffer is empty. At buffer termination, the system ensures that the buffer is printed and the paper is spaced upward one line.

During non-VFC write image operations, the user buffer is output exactly as it exists in memory. The system does not ensure that the data is printed or that the paper is properly moved. The user should be familiar with the characteristics of the particular device being used.

For information on I/O operations with VFC, see Section 5.2.

5.9 TAPE CASSETTE

Variable length records are supported by Perkin-Elmer tape cassettes. During input, ASCII, binary, and image modes are identical. Data is read from the cassette into the user buffer. The transfer terminates when the buffer is full or at end of record, whichever comes first. If the record is longer than the buffer, error status is not returned. Parity errors in the unread part of the record can be detected. If a parity error occurs, five retries are attempted before error status is returned. When a parity error status is returned, the tape is positioned in the interrecord gap following the record in error.

During output, ASCII, binary, and image modes are identical. Data is written from the user buffer until the buffer is empty. The system retries five times on parity errors.

The driver generates an end of tape condition, whether the tape is positioned at the beginning or at the end of the reel. It must be assumed from the last operation what position end of tape is actually referring to.

Since the two drives on an intertape cassette share logic, only one drive of a cassette pair (e.g., X'45' and X'55') can be active at a time.

Continuous mode operations are used to pass requests to the driver within the time required (10 milliseconds for read; 30 milliseconds for backspace).

5.10 MAGNETIC TAPE

Data transfer operations can be performed in standard and gapless I/O format.

5.10.1 Standard Input/Output (I/O)

Variable length records are supported by Perkin-Elmer magnetic tape devices. During input, data is read into the user buffer from the magnetic tape. The transfer ends on buffer full or end of record, whichever comes first. If a parity error occurs, the driver retries the read operation before an error status is returned. The number of retries performed is determined by the retry value set in the DCB or specified by the user in the SVC 1 extended function code field for data transfer operations. After a parity error occurs during a read forward operation, the tape is positioned in the interrecord gap preceding the record with the error. If an error occurs during a read backward operation, the tape is physically positioned following the record with the error.

During output, data is written from the user buffer to the magnetic tape until the buffer is empty. On parity error, the tape is positioned before the record causing the error, the record gap is extended and the write operation is retried. Again, the number of retries is determined by the retry value set in the DCB or specified in the SVC 1 parameter block.

In addition to giving users control over the number of retries for data transfer errors, OS/32 provides the ability to erase a variable length of tape and to select the recording density, via SVC 1 and SVC 7, respectively. See the OS/32 Supervisor Call (SVC) Reference Manual for more information on how to implement these features.

For read and write requests, ASCII, binary, and image requests are identical.

The minimum number of bytes that can be transferred by a tape drive is four. All data transfers must start on a halfword (i.e., even byte) boundary and should specify an even number of bytes (i.e., end address odd).

On a read operation, end of tape may be detected on a different record than on a write operation because of mechanical tape positioning. If rewind is issued at beginning of tape, the driver returns normal status. Ensure that the tape is loaded at beginning of tape unless some other condition is expected.

5.10.2 Gapless Input/Output (I/O)

Data transfer operations in gapless mode consist of a task reading or writing data buffers to a magnetic tape with no intervening interrecord gaps, using only one SVC 1. To perform gapless I/O to a magnetic tape, a task must issue an SVC 1 call that specifies, among other things, a pair of buffer queues, the IN-QUEUE and the OUT-QUEUE. The driver takes buffers from IN-QUEUE and returns used buffers to the OUT-QUEUE. The task processes the buffers from the OUT-QUEUE and returns these buffers to the IN-QUEUE for reuse by the driver. A special gapless format SVC 1 parameter block must be used for gapless I/O operations. Buffers used within a single gapless I/O operation must be equal in length, with the possible exception of the last buffer used.

5.11 DISK STORAGE

Perkin-Elmer disk devices support variable length records. During input, a current sector pointer is maintained. On a sequential read, data is read into the user buffer from the disk, starting at the current sector, until the buffer is full. If an attempt is made to read beyond the end of the disk, end of medium (EOM) status is returned. On a random read request, data is read from the disk starting at the sector specified by the random sector address passed with the request, until the buffer is full. If an attempt is made to read beyond the end of the disk, EOM status is returned with data transferred. ASCII, binary, and image requests are identically treated.

During output, data is written from the user buffer to the disk, starting at the current sector (for sequential writes) or at the specified sector (for random writes), until the buffer is empty. Attempts to write past the end of the disk cause EOM status to be returned. In this case, no data is transferred.

Errors on data transfers cause the operation to be retried several times before returning error status.

All data transfers start on a sector boundary, but can end on any byte of a sector. If the size of the user buffer is less than the record size of an indexed file to which it is written, the remaining bytes will be filled with blanks for ASCII writes or binary zeros for binary writes. If a record written to a contiguous, extendable contiguous, or nonbuffered indexed file is less than the file's record length, the last byte or two bytes are propagated through the remaining unfilled bytes of the last 256-byte sector of the record.

Only e-tasks and privileged u-tasks and d-tasks linked with bare disk privileges (OPTION DISC) can access a bare disk. Non-privileged u-tasks and d-tasks access the disk via the contiguous, extendable contiguous, nonbuffered indexed, or indexed file handlers.

5.12 FLOPPY DISK

Variable length records are supported by Perkin-Elmer floppy disks. During input, a current sector pointer is maintained. On a sequential read, data is read from the disk starting at the current sector into the user buffer until the buffer is full. On a random request, the data is read from the disk starting at the sector specified by the random sector address passed with the request, until the buffer is full. If an attempt is made to read beyond the end of disk, EOM status is returned with data transferred. ASCII, binary, and image requests are identically treated.

During output, data is written from the user buffer to the disk, starting at the current sector pointer (for sequential writes) or at the specified sector (for random writes), until the buffer is empty. If an attempt is made to write beyond the end of the disk, EOM status is returned with no data transferred. ASCII, binary, and image requests are identically treated.

Errors on data transfers cause the operation to be retried ten times before returning error status.

All data transfers start on a logical 256-byte sector boundary (two physical sectors on the floppy). Transfer can end on any byte of a sector.

The floppy disk driver is designed for use by the file manager. A user program cannot access a bare disk unless it is an e-task or privileged u-task or d-task linked with bare disk privileges (OPTION DISC). For nonprivileged u-tasks and d-tasks, the disk is accessed by the contiguous, extendable contiguous, nonbuffered indexed, or indexed file handlers.

5.13 VIDEO DISPLAY UNIT (VDU) TERMINALS

Variable length records are supported by all Perkin-Elmer VDU terminals.

During non-VFC read ASCII operations, data read is masked to 7-bit ASCII. Data is read until the buffer is full or a carriage return is encountered, whichever occurs first. Upon termination, a CR/LF sequence is sent to the screen.

During non-VFC write ASCII operations, the buffer is scanned to eliminate trailing blanks. Data is then sent to the VDU until the buffer is exhausted, the last nonblank character has been processed, or until a carriage return is found in the data stream. A line feed is automatically appended to the detected carriage return; or, if no carriage return is detected, an LF/CR sequence is sent to the terminal.

During non-VFC image I/O, none of the above formatting actions occur. The amount of data requested is output or read in without masking to 7-bit ASCII, eliminating trailing blanks, checking for control characters, or detecting or appending carriage returns or line feeds. On image read, however, an ASCII CR is detected as an end of line sentinel.

For information on I/O operations with VFC, see Section 5.2. See the OS/32 Operator Reference Manual for an explanation of the function control keys available on Perkin-Elmer VDU terminals.

5.14 8-LINE INTERRUPT MODULE

Interrupt simulation (SINT) is the only attribute supported by the Perkin-Elmer 8-line interrupt module. The module provides the processor with eight interrupt lines from external equipment and acknowledges interrupts on a priority basis. Any line can be selectively enabled or disabled. Several lines can be concurrently enabled. An interrupt does not transfer any data, nor is any status given.

5.15 DIGITAL MULTIPLEXOR

ASCII operations are not supported by the Perkin-Elmer digital multiplexor. During input, the second byte of the random address field contains the segment and point number to be read. Data is read from the point specified until the buffer specified by the starting and ending address is full.

During output, the second byte of the random address field contains the segment and point number to be written to. Data is written until the buffer specified by the starting and ending address is exhausted.

5.16 CONVERSION EQUIPMENT

The analog conversion equipment used with Perkin-Elmer 32-bit computers cannot be programmed in the device independent manner of other peripheral devices. The chassis, channel and card addresses, and data values are directly passed to the real-time analog system controller as the 16-bit words that are obtained from the user.

During input, the random address field of the SVC 1 parameter block contains the starting address of a table containing analog-to-digital converter addresses (chassis address, channel address, and card address). The user buffer, which the start and end addresses of the parameter block determine, is loaded with the digitized data obtained from these analog-to-digital converters.

The table length containing the converter addresses is equal to the length of the buffer. It is the user's responsibility to provide valid addresses. Since the analog input system mode of the controller is used for READ, if a nonexistent chassis is addressed, zero data is stored and no other indication is given.

During output, the user buffer is assumed to contain sequential pairs of alternating digital-to-analog converter addresses and the corresponding data to be converted; i.e., ADD1, DATA1, ADD2, DATA2,........ADDn, DATAn. The address and data are directly passed to the real-time analog system controller.

The control output mode of the controller is used for write operations. If a nonexistent chassis is addressed, the status is set to X'88', and the remainder of the I/O is aborted.

Each write sequence to any converter must consist of two halfwords. One halfword specifies the adapter to do the conversion; the other halfword contains the data to be converted. A buffer must be a multiple of two halfwords in length; otherwise, any attempt to do a write results in a memory fault.

For read and write operations, ASCII/binary and image/format requests are identical.

5.17 ANALOG INPUT CONTROLLER

Variable record lengths are supported by the Perkin-Elmer analog input controller. ASCII operations are not supported. Command functions are ignored.

The random address field of the supervisor call 1 (SVC 1) parameter block contains the gain and address of the first channel to be sampled. The format is shown in Figure 5-1. Dividing the length of the user buffer (END-START+1) by two determines the number of channels to sample. The digitized data is sequentially stored in the user buffer, one halfword per channel.

```
 ---------------------------------------------------------
|    Gain    |                             |   Address    |
 ---------------------------------------------------------
Bits:
 0          3 4                          11 12          15
```

Figure 5-1 Random Field Format

The driver accepts only random calls, meaning that the first address is selected at random and that further addresses are sequential (in the same call). The start address must be on an even address boundary and the end address must be on an odd address boundary, since the analog input controller is a halfword device. This complies with the Instrument Society of America (ISA) definition of sequential analog input.

5.18 ANALOG OUTPUT CONTROLLER

All command functions are ignored by the Perkin-Elmer analog output controller. One halfword of data is obtained from the user buffer in the format specified in Figure 5-2 and written to the device for conversion. This procedure is repeated until all halfwords in the user buffer are output. Dividing the length of the user buffer (END-START+1) by two computes the number of halfwords to be output.

```
 ------------------------------------------------
|           |                        |   Data    |
 ------------------------------------------------
Bits:
 0         3 4                     11 12        15
```

Figure 5-2 Analog Output Data Format

Binary image is treated identically to binary formatted; the image bit is ignored. The sequential/random bit is also ignored. The start address must be aligned on an even boundary, whereas the end address must be on an odd boundary because the analog output controller is a halfword device.

5.19 DIGITAL INPUT/OUTPUT (DIO) CONTROLLER

All command functions are ignored by the Perkin-Elmer DIO controller. The number of transfers is computed using the start and end address fields: (END-START+1)/2. Resetting the sequential/random bit in the function code field causes transfers to occur sequentially without interruption. This is a nonhandshaking transfer mode. In the handshaking transfer mode, the sequential/random bit is set, and each transfer occurs only after the internal strobe line is pulsed. A timeout rate for each transfer is set at a constant of four seconds.

During a binary read operation, one halfword of data is transferred to the user buffer whose starting address is stored in the SVC 1 parameter block. Each halfword of data from subsequent binary read operations is stored sequentially in the user buffer.

For binary write operations, the buffer starting address in the SVC 1 parameter block points to a buffer (K1) consisting of image halfwords for transfer to an output device. The random address field of the SVC 1 parameter block points to another buffer (K2) of halfwords designating masks, each of which defines the corresponding bit position of the halfwords in K1 that is to be changed.

The length of K2 must be the same as that of K1. A bit set in K2 indicates that the digital output is changed to the state defined by the corresponding bit position in K1. The following logical expression computes the halfwords transferred to the digital output card:

$$(K1 \circ \overline{K2}) + (\overline{K2} \circ R)$$

Where:

$\bullet$	means logical AND.
$\oplus$	means logical OR.
$\overline{K2}$	means one's complement of K2.
R	is the last known content of the output register.

Binary image is treated identically to binary formatted; the drive ignores the formatted/image bit of the SVC 1 function code. Both handshaking and nonhandshaking transfer modes are supported. The start and random addresses of the user buffer must be aligned to even boundaries, whereas the end address must be aligned to an odd boundary because the digital I/O controller is a halfword device.

CHAPTER 6
PROGRAMMING IN A MODEL 3200MPS SYSTEM MULTIPROCESSING ENVIRONMENT

6.1 INTRODUCTION

Programming in a Model 3200MPS System multiprocessing environment is similar to programming in a uniprocessing environment. However, due to the nature of the hardware configuration, the Model 3200MPS System environment offers one major programming advantage, increased throughput. For efficient use of this expanded computing ability, the system level programmer should take the following into consideration:

- the selection of tasks that are to run on each processor in the system at any given time,
- the assignment of processor units to the selected tasks,
- the preparation of the auxiliary processing units (APUs) for task execution,
- the rescheduling of tasks from one processor to another in response to a failure condition or processor overload,
- the prevention of invalid data variables caused when two tasks, running on different processors concurrently, read and modify a common data structure, and
- the measurement of real-time performance of the individual tasks in the system.

This chapter focuses on some techniques that can be used by an assembly language programmer in solving some of the programming problems that are unique to the Model 3200MPS System multiprocessing environment.

6.2 DESIGNING TASKS TO RUN ON A MULTIPROCESSING SYSTEM

The main performance advantage of designing an application to run on a multiprocessing system is that a job can be broken down into several parts that can be run on different processors simultaneously. For example, a job can be divided among a number of tasks that control individual operations, such as process input/output (I/O), perform calculations resulting from a particular action, and provide an operator interface for reporting and responding to the results of the calculations.

The individual APUs running these tasks can transmit all status information regarding the components of the system to another task, called the supervisor monitor. The supervisor monitor can then output messages to a console or printer as the status is received. Another function of the supervisor monitor could be to store a code in a status word in memory that can be accessed by a standby task. The standby task then would be able to periodically check the status of the system and adjust task execution accordingly.

Once the programmer has divided a job into several tasks that can be run simultaneously, the next step should be to assign each task to a processor for execution. It should be remembered that execution of a compute-intensive task on an APU increases overall system performance, while an I/O intensive task running on an APU decreases system performance. Because the operating system executes exclusively on the central processing unit (CPU), each I/O request made by an APU task causes the task's execution to be transferred back to the CPU for operating system suppport. Hence, all I/O intensive tasks should be assigned to the CPU for execution.

6.3 ASSIGNING A TASK TO A PROCESSOR

OS/32 supports a multiprocessing configuration consisting of one CPU and one to nine APUs. Each processor is given a unique identifying number. By definition, the identifying number for the CPU is zero. APUs are numbered from one through nine.

The operating system also defines a set of logical processing units (LPUs). An LPU is mapped to an APU or to the CPU. An APU can be mapped to more than one LPU, but each LPU can be mapped to one and only one APU. The identification number of the processor to which each LPU is mapped is contained in a system data structure known as the logical processor mapping table (LPMT). Before a task can run on an APU, the task must be assigned to an LPU that is mapped to the processor. (If the LPU is mapped to APU 0, the task will execute on the CPU.)

As in a uniprocessor system, a task can load another task for execution on an APU via supervisor call 6 (SVC 6). Unlike a uniprocessor system, however, the programmer must consider two additional SVC 6 function codes, assign LPU (SFUN.LPM) and transfer to LPU (SFUN.XLM). These function codes assign the task to an LPU and transfer execution from the CPU to the APU mapped to the assigned LPU.

The following example shows how to code an SVC 6 parameter block for sending a task to the APU assigned to LPU 2. It is assumed that logical unit 5 (lu5) has been assigned to the file DIRECTED.TSK (via the OS/32 ASSIGN command or an SVC 7) and is positioned to the first byte of the LIB for the task, and that LPU 2 has been mapped to an APU.

```
*****************************************************************
*                                                               *
*         This example loads and starts a copy of a task to    *
*         run on an APU in a Model 3200MPS System.             *
*                                                               *
*****************************************************************
          $SVC6
          ALIGN 4
PARBLK    DS    SVC6                ALLOCATE STORAGE FOR PARBLK
ENDBLK    EQU   *
*SET LOAD, ASSIGN LPU, LPU-DIRECTED, & START FUNC CODES
          ORG   PARBLK+SVC6.FUN
          DC    SFUN.DOM!SFUN.LM!SFUN.LPM!SRUN.XLM!SFUN.SIM
          ORG   PARBLK+SVC6.LU
          DB    5                   LU OF DIRECTED.TSK (IMAGE)
          ORG   PARBLK+SVC6.SAD
          DC    0                   TASK EXECUTION START ADDR
          ORG   PARBLK+SVC6.SOP
          DC    0                   START OPTIONS (none)
          ORG   PARBLK+SVC6.SEG
          DC    Y'40'               TASK WORKSPACE
          ORG   ENDBLK
START     EQU   *
*SETUP NAME OF TASK TO BE LOADED
          LI    R1,CAPU1
          ST    R1,PARBLK
          LI    R1,C'TASK'
          ST    R1,PARBLK+4
*ASSIGN LPU NUMBER
          LIS   R1,2
          STB   R1,PARBLK+SVC6.LPU
*ISSUE SVC6 TO LOAD TASK FROM LU5
          SVC   6,PARBLK
          END   START
```

After the SVC 6 in the above example is executed, the task will be loaded into memory from the file (DIRECTED.TSK) with a workspace of 640 (X'40') bytes. When the task is loaded, the task manager dispatches it to the APU mapped into APU 2.

NOTE

If the LPU-directed function code (SFUN.XLM) was not set, the task is executed on the CPU. If the task was linked with the APU-only (APUONLY) task option, the task is not dispatched to an APU and pauses.

For more information on SVC 6, see the OS/32 Supervisor Call (SVC) Reference Manual.

6.4 PREPARING AN AUXILIARY PROCESSING UNIT (APU) FOR TASK EXECUTION

Even though a task is assigned to an LPU mapped to an APU, the task will not be dispatched to the APU unless the APU is in a valid task operating state. OS/32 maintains four operating states for an APU, each differing in the degree of APU readiness for task execution. These states are:

- DISABLED - APU is unavailable for all purposes except running the power up link check procedure. (An APU is placed in the DISABLED state upon loading of the operating system or power restore.)
- OFF (ENABLED) - APU has successfully passed the power up link check procedure but is not available for task execution.
- ON EXCLUSIVE - APU is available for scheduling and executing a single designated task only.
- ON - APU is fully available for scheduling and executing any task in the system.

Figure 6-1 shows the valid APU operating states. A task can control the task operating state of an APU by executing the corresponding SVC 13 APU control and mapping functions. The SVC 13 APU control functions are used by a task to enable or disable an APU. The SVC 13 mapping functions are used to mark the APU on for task scheduling and to map the APU to a particular LPU number in the LPMT.

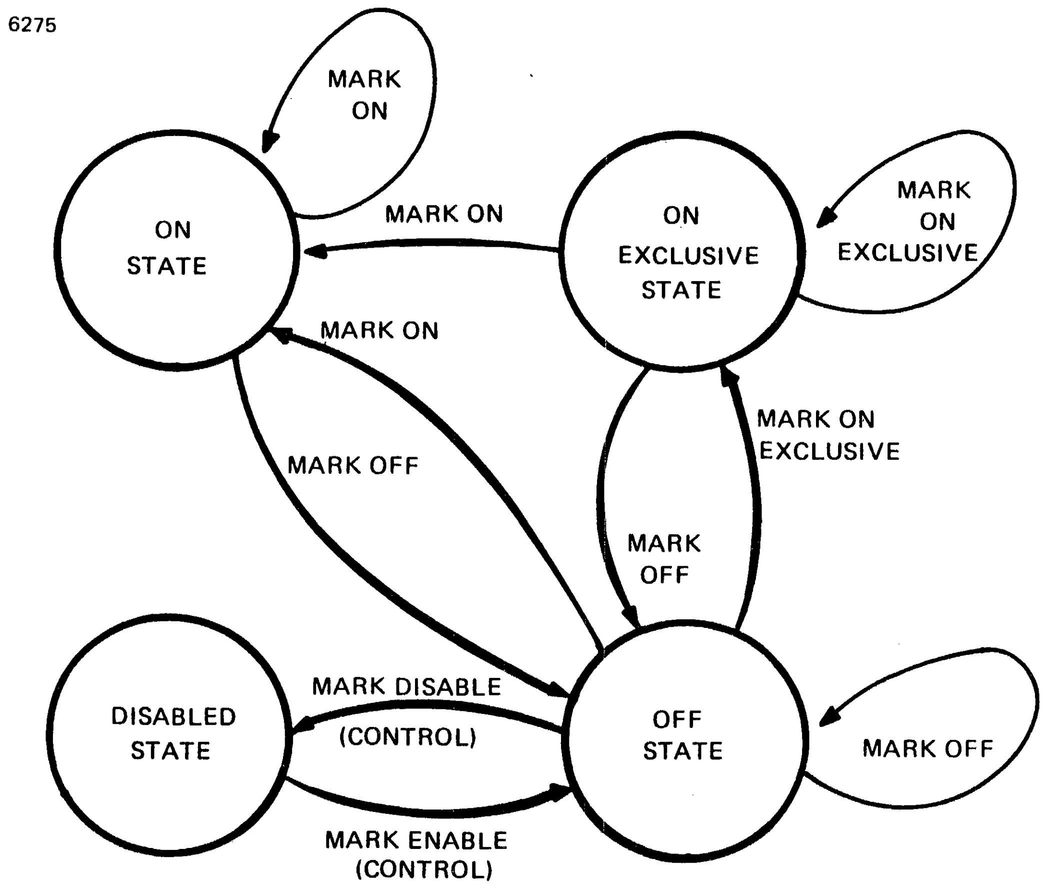


Figure 6-1 Valid APU Operating States

The following code demonstrates how SVC 13 is used to enable an APU and mark it on for task scheduling. This example does not check for SVC 13 execution errors.

Example:

```
          $SVC13
          ALIGN 4
ENABLE    DS    SVC13     ALLOCATE STORAGE FOR SVC 13 PARBLK
ENABLEE   EQU   *
*GAIN CONTROL RIGHTS, ENABLE APU, SEND APU COMMAND, AND
*RELEASE CONTROL RIGHTS
          ORG   ENABLE+SV13.OPT
          DB    X'C9'
          ORG   ENABLE+SV13.FUN
          DB    X'03'               FUNCTION CODE=3
          ORG   ENABLE+SV13.DOP
          DB    X'01'               SEND START APU COMMAND
          ORG   ENABLE+SV13.APN
```

```
            DB     2                     APU NUMBER
            ORG    ENABLED
***BUILD SVC 13 PARAMETER BLOCK FOR MAPPING APU
            ALIGN 4
MAP         DS     SVC13.                ALLOCATE STORAGE FOR SVC 13 PARBLK
MAPE        EQU    *
*GAIN MAPPING RIGHTS, MARK APU ON, MAP APU INTO LPMT,
*RELEASE MAPPING RIGHTS
            ORG    MAP+SV13.OPT
            DB     X'B1'
            ORG    MAP+SV13.FUN
            DB     2                     FUNCTION CODE=2
            ORG    MAP+SV13.DOP
            DB     2                     LPU NUMBER TO BE MAPPED
            ORG    MAP+SV13.APN
            DB     2                     APU TO MAP LPU TO
            ORG    MAPE
******ISSUE SVC 13 TO ENABLE AND MAP APU*******
            SVC    13,ENABLE             ENABLE APU
            SVC    13,MAP                MAP APU TO LPU 2
```

6.5 TASK RESCHEDULING

In a uniprocessor system, priority scheduling determines the execution flow of the tasks in the system. In order to affect task scheduling, a programmer must change the priority of the tasks in the system. In an OS/32 Model 3200MPS multiprocessing system, this is only true for tasks that are running on the CPU. Rescheduling tasks on an APU is accomplished via the SVC 13 control function options, as described in the following sections.

6.5.1 Monitoring and Preempting Auxiliary Processing Unit (APU) Task Execution

This section will examine the methods used by an APU monitor task to:

- receive status signals from an APU, and
- preempt the current task executing on an APU with another task after a certain time interval has elapsed.

To receive a status signal from an APU, the APU must be connected and thawed (via SVC 6) as a trap-generating device to the monitor task. In addition, the monitor task must have the appropriate bits in its task status word (TSW) set, a task queue to receive the status signal, and a task queue trap handling routine to service the trap. The following example demonstrates how to code a typical APU monitor program to receive and handle task queue traps from an APU. For more information on task trap handling see the OS/32 Application Level Programmer Reference Manual.

Example:

```
****  Define a task queue to receive APU signals  ***************
*                                                               *
         ALIGN 4
   TASKQ DLIST 100  DEFINE TASK Q OF 100 ELEMENTS.
*                                                               *
* Put the address of task queue in UDL (UDL.TSKQ)               *
*                                                               *
         LA    R14,TASKQ
         ST    R14,UDL.TSKQ
*                                                               *
* Set TSW bits to enable the applicable task traps.             *
*                                                               *
         LI    R14,TSW.TSKM+TSW.APTM
*                                                               *
* TSW.TSKM enables task queue service traps                     *
* TSW.APTM enables signals from APU                             *
* Save TSW values to enable APU signals and task Q entries      *
         ST    R14,ENTRIES        SAVE TSW VALUES
* TO ENABLE APU SIGNALS AND TASK Q ENTRIES
*                                                               *
* SET UP TSW FOR TRAPS IN UDL                                   *
*                                                               *
         LA    R15,QSERVICE
         STM   R14,UDL.TSKN       SET UP TSW ON TRAPS IN UDL
         SVC   9                  ENABLE TASK QUEUE ENTRIES
```

For information on writing a task queue trap handling routine that removes the APU status entries from the task queue, see the OS/32 Application Level Programmer Reference Manual.

The following code demonstrates a method of connecting the APUs in a system as trap-generating devices to the APU monitor task.

Example:

```
* Enable each APU in the system if it is not enabled and then   *
* connect to each APU, but first                                *
* fetch a copy of the LPMT to obtain the                        *
* number of APUs in the system.                                 *
*                                                               *
START    SVC   13,LPMTFET
         LB    R1,BUFFER1+1       LOAD MAX APU NO.INTO R1
* SET UP SVC 13 PARAMETER BLOCK TO                              *
* FETCH APU STATUS                                              *
         LIS   R3,1
         STB   R3,FETAPU+SV13.FUN SET UP FUNCTION CODE 1
         LA    R4,APUBUF                                        *
         ST    R4,FETAPU+SV13.BUF SET UP BUFFER ADDR.           *
```

```
          LHI   R3,256
          STH   R3,FETAPU+SV13.LEN  SET UP BUFFER LENGTH
*                                                              *
* SET UP SVC13 PARAMETER BLOCK TO ENABLE THE APU               *
*                                                              *
          LIS   R3,3                SET UP SVC 13 FUNC CODE 3
          STB   R3,ENABAPU+SV13.FUN
          LIS   R3,X'C1'            SET UP CONTROL OPTIONS
          STB   R3,ENABAPU+SV13.OPT GAIN,ENABLE,RELEASE
*                                                              *
* GET THE APU STATUS.  IF APU IS DISABLED,                     *
* ATTEMPT TO ENABLE IT.  IF APU CAN'T BE                       *
* ENABLED, LOG MESSAGE TO CONSOLE AND                          *
* CONNECT TO IT ANYWAY JUST IN CASE IT IS                      *
* ENABLED AND MARKED ON LATER.                                 *
*                                                              *
APULOOP   EQU   *
* GET APU STATUS
          STB   R1,FETAPU+SV13.APN  SET UP APU NO.
          SVC   13,FETAPU           ISSUE SVC 13
          LH    R4,FETAPU+SV13.ERR  GET SVC 13 ERROR STATUS
          BZ    GETSTAT             IF NO ERROR-GET APU STATUS
          BNE   ER.ROUTE            IF ERROR, BRANCH TO ER.ROUT
          LB    R5,APUBUF+5         GET 2ND BYTE OF APU S-STATUS
          BNZ   CONNECT             NOT DISABLED, GO CONNECT
*APU IS DISABLED; ISSUE SVC 13 TO ENABLE IT.
          STB   R1,ENABAPU+SV13.APU SAVE APU NUMBER
          SVC   13,ENABAPU          ENABLE THE APU
          LH    R3,ENABAPU+SV13.ERR GET SVC 13 ERROR STATUS
          BNZ   ENAB.ERR            BRANCH TO ERROR ROUTINE ON ERROR
CONNECT   EQU   *
*                                                              *
*SAVE APU NO. AS PART OF APU'S TGD MNEMONIC                    *
          STB   R1,SVC6.DEV
*ISSUE SVC6 TO CONNECT AND THAW THE APU                        *
*                                                              *
          SVC   6,APUTRAPS
          LH    R6,SVC6.STA         GET SVC 6 ERROR STATUS
          BZ    NEXT.APU            NO ERROR-GO CONNECT TO NEXT
*                                   APU
          STB   R1,CONB.ERR+24      SAVE APU NO. IN MESSAGE
          SVC   2,LOGMSG            LOG MESSAGE: COULD NOT
*                                   CONNECT TO APUX
NEXT.APU  SIS   R1,1                MOVE ON TO NEXT LOWEST APU NO.
          BP    APULOOP             GO HANDLE NEXT APU.
```

The parameter blocks used in the above example are defined as follows:

```
*SVC 13 FETCH LPMT Parameter Block and Buffer
          ALIGN 4
LPMTFET   DS    SVC13
ENDPBK    EQU   *
          ORG   LPMTFET+SV13.FUN
          DB      X'00'                  SET FUNC CODE 0
          ORG     LPMTFET+SV13.BUF
          DAC     BUFFER 1               DATA BUFFER ADDR
          ORG     LPMTFET+SV13.LEN
          DC      H'50'                  MAX LENGTH OF BUFF
          ORG     ENDPBK
          ALIGN 4
BUFFER    DS      50
*                                                                 *
* SVC13 Fetch APU Status Parameter Block & Buffer                 *
*         ALIGN 4                                                 *
* FETAPU DS       SVC13

          ALIGN 4
APUBUF    DS      256
*
*** SVC13 Enable APU Parameter Block
*
          ALIGN 4
ENABAPU   DS      SVC13.
*                                                                 *
* SVC 6 Connect & Thaw APU Parameter Block                        *
*                                                                 *
          $SVC6
          ALIGN 4
APUTRAPS DS       SVC6.
ENDAPUTB EQU      *
          ORG     APUTRAPS+SVC6.FUN
          DS      Y'C000 C000'           SVC6 FUNC CODE-
*                                        SELF-DIRECTED, CONNECT & THAW
*
          ORG     APUTRAPS+SVC6.DEV
          DC      C'APU'                 TRAP-GENERATING DEVICE MNEMONIC
          ORG     ENDAPUTB
*
**** SVC 2 Log Message Parameter Block
*
LOGMSG    DB      0,7
          DCZ     CONE.ERR-CONB.ERR
CONB.ERR DB       C'UNABLE TO CONNECT TO APU'
CONE.ERR EQU      *
```

The code in the above example allows the monitor to receive traps from the APUs. Status returned from these traps can be reported to the console (via SVC 1 or SVC 2 Code 7) or to a file designated for the APU output (via SVC 1). In addition, this monitor program could be coded to run a certain task (TASK1) every 10 minutes on a specific APU. To do this, the monitor sets an interval timer via SVC 2 Code 23. Upon expiration of the timer, the monitor task issues an SVC 13 Code 3 to preempt the current executing task on the APU, as shown below.

Example:

```
                    SVC    13,PREQ
                    .
                    .
                    .
                    ALIGN 4
        * PREEMPT QUEUE, EXECUTE CONTROL COMMAND, AND RELEASE CONTROL RIGHTS
        PREQ.OPT DB     X'B9'                   SET SVC 13 OPTIONS:
        PREQ.FUN DB     X'03'                   SET FUNCTION CODE-
        *                                       CONTROL FUNCTION
        PREQ.DOP DB     X'01'                   DIRECTIVE OPTION-
        *                                       START APU
        PREQ.APN DB     X'01'                   APU NO. - APU 1
        PREQ.APS DS     2                       APU HARDWARE STATUS
        PREQ.ERR DS     2                       SVC 13 ERROR STATUS
        PREQ.BUF DAC    BUF2                    DATA BUFFER ADDRESS
        PREQ.USE DS     2                       LENGTH OF BUFFER USED
        PREQ.LEN DC     H'8'                    MAX LENGTH OF BUFFER
                 ALIGN  4
        BUF2     DC     C'TASK1 ƀƀƀ'            TASK ID BUFFER
```

Execution of the above SVC 13 will cause the monitor to gain control rights to the specified APU (APU 1), provided that the task has been link edited with the APCONTROL task option and no other task has control rights to the APU. The control options, specified in the SVC 13 parameter block, will then cause the following actions:

- Execution of the current executing task on the APU will be stopped.
- The current task will be rescheduled to the end of the APU ready queue.
- The APU's ready queue pointer will be repositioned to point to TASK1. (This will cause TASK1 to be selected as the next task to be executed on the APU.)
- The APU will be restarted for execution of TASK1.
- The monitor task will release the control rights to the APU.

The remaining code in the monitor program should check the PREQ.ERR field of the PREQ parameter block for errors as follows:

Example:

```
    LH     R2,SV13.ERR
    BNZ    ERR.PREQ
```

If an error has occurred, ERR.PREQ can log a message to the console.

Finally, to reexecute TASK1 in 10 minutes, the interval timer (via SVC 2 Code 23) should be reset so that the SVC 13 Code 3 to preempt the current APU task can be reissued when 10 minutes have elapsed.

See Chapter 3 for more information on SVC 13. See the OS/32 Supervisor Call (SVC) Reference Manual for more information on SVC 6 and SVC 2 Code 23.

6.5.2 Verifying Task Transfer to an Auxiliary Processing Unit (APU)

It may be necessary for a task to verify whether or not it has actually been transferred to an APU for processing. For example, suppose a task on the CPU is assigned to LPU 3 and executes the following instruction:

```
    RSCH  R1,2
```

Execution of this instruction will cause OS/32 to set the LPU directed status of the task. The OS/32 task manager will then attempt to transfer the task to the APU mapped to LPU 3. Suppose LPU 3 is mapped to APU 3. To verify that the task has indeed been transferred to APU 3, the next instructions executed by the task could be:

```
             LIS   R1,0                   Get RTSM PULSE LINE
    * TO PULSE
             LI    R2,15                  FILL IN APU ID
             GSIG  R1,R2                  GENERATE SIGNAL
    * HERE THE NO. (15) CAN NEVER MATCH
    * THE APU ID IN THE RTSM.  NO SIGNAL WILL BE
    * SENT.  INSTEAD, ONLY THE APU ID IS RETURNED TO R1
```

After execution of GSIG, R1 will contain the number of the APU on which the task is currently executing. See the Model 3200MPS System Instruction Set Reference Manual for more information on the RSCH and GSIG instructions.

6.5.3 Transferring a Task from an Auxiliary Processing Unit (APU) to the Central Processing Unit (CPU)

Under certain conditions, a monitor task may need to transfer some other task back to the CPU for processing. The task to be transferred may be executing on an APU or waiting on its ready queue. The monitor task can transfer a task back to the CPU by issuing an SVC 6, specifying the following function codes:

- Suspend (SFUN.SM)
- Set CPU directed (SFUN.XCM)
- Release (SFUN.RM)

The suspend will transfer the task back to the CPU by setting the CPU directed flag. Upon release, it will stay on the CPU and not be dispatched back to the LPU.

Example:

```
             SVC    6,CPUDIR
                .
                .
                .
             ALIGN 4
             $SVC6
    CPUDIR   DS     SVC6.
    CPUDIRE  EQU    *
             ORG    CPUDIR+SVC6.ID
             DC     C'TASK1ɓɓɓ'          ID OF TASK TO BE
    *                                         TRANSFERRED
             ORG    CPUDIR+SVC6.FUN
    * SET SUSPEND,CPU-DIRECTED, & RELEASE FUNC CODES
    * FOR TASK1
             DB     SFUN.DOM!SFUN.SM!SFUN.XCM!SFUN.RM
             ORG    CPUDIRE
```

Execution of this SVC 6 causes TASK1 to be suspended (if it is not already in a wait state) and transferred to the CPU. Setting the CPU directed bit in the task's TSW directs the task manager to ignore its LPU assignment and to schedule this task for execution on the CPU. When released, the task will execute on the CPU at the location following the instruction that was executed before the task was suspended. If the SVC 6 in the above example did not set the CPU directed bit in the task's TSW, the task will again be dispatched to the APU assigned to its LPU number upon release from the suspended state.

6.6 PREVENTING MEMORY ACCESS CONFLICTS

When several processors are executing simultaneously, it is possible for tasks running on two or more processors to need access to the same data. For example, suppose two tasks share a buffer list consisting of 30 buffers defined as follows:

```
BLISTBIT DS    2
BUFLIST  DLIST 30
```

BUFLIST is a list containing the addresses of the buffers. BUFLIST, and the actual buffers, reside in an area of memory shared by the two tasks. One task collects data, writes it to a buffer and adds the address of that buffer to the bottom of the list. The other task removes an address of a buffer from the top of the list and processes it. Since in a Model 3200MPS System both tasks may be running simultaneously on different APUs, both tasks may attempt to access the list at the same time. The Test and Set instruction (TS) can be used to ensure that only one task can access the buffer at a time.

To ensure that only one task can access BUFLIST at a time, a test and set operation is performed on BLISTBIT. BLISTBIT acts as a lock out mechanism that is set and reset. A task can only access BUFLIST if BLISTBIT is not set.

6.6.1 Avoiding System Deadlock

When using the test and set operation, care should be taken to ensure that system deadlock is avoided.

For example, suppose task A uses TS to lock out data structure X while task B is locking out data structure Y. Task A now finds that it needs to access data structure Y, so it waits for Y to be released. Similarly, Task B finds it needs to access data structure X, so it waits for X to be released. Since each task holds the data structure needed by the other, processing stops. Both tasks are deadlocked.

To avoid system deadlock, the test and set instruction should be used with a timeout mechanism. The following example shows how to prevent memory access conflicts without system deadlock.

Example:

```
         TS     BLISTBIT              TASK CHECKS IF IT CAN GET
*                                     ACCESS TO LIST
         BNM    CONTINUE              PROCESS LIST IF FREE
         LI     R2,50                 LOAD TIMEOUT VALUE OF 50
*                                     MICROSEC IN R2
SETBITLP EQU    *                     TIMER ROUTINE
         SIS    R2,1                  DECREMENT TIMEOUT COUNT
         BM     TIMEOUT               BRANCH TO TIMEOUT ROUTINE
*IF BRANCH TO TIMEOUT IS TAKEN IT MEANS THAT THE                *
*TASK STILL COULD NOT GET ACCESS TO LIST                        *
*THE TIMEOUT ROUTINE PRINTS A MESSAGE TO THE CONSOLE            *
*SO OPERATOR CAN TAKE NECESSARY ACTION                          *
*ELSE CONTINUE                                                  *
         LH     R4,BLISTBIT           USE APU CACHE TO MATCH LOCKS
         BMS    SETBITLP              BUFLIST NOT AVAILABLE YET;
*                                     TRY AGAIN
         TS     BLISTBIT              BUFLIST IS AVAILABLE SO
*                                     ATTEMPT TO GRAB ACCESS
         BMS    SETBITLP              NOT QUICK ENOUGH, RETRY
**IF SUCCESSFUL, PROCESS LIST                                  **
*                                                               *
CONTINUE EQU    *
*        .                                                      *
*        .                                                      *
*        .                                                      *
*ACCESS BUFLIST EITHER BY ABL (ADD TO BOTTOM OF LIST            *
*INSTRUCTION) OR  RTL (REMOVE FROM TOP OF LIST INSTRUCTION).    *
*        .                                                      *
*        .                                                      *
*        .                                                      *
*AFTER PROCESSING BUFFER, UNLOCK BLISTBIT SO OTHER TASK CAN     *
*ACCESS IT.                                                     *
*                                                               *
         LIS    R4,0
         RBT    R4,BLISTBIT
```

6.7 MEASURING REAL-TIME PERFORMANCE ON A MODEL 3200MPS SYSTEM

The OS/32 system macro library provides a set of timer macros that can be used to measure the real-time performance of individual tasks running in a Model 3200MPS System. These macros allow the programmer to set up a named timer in memory. A named timer can be compared to a stopwatch that measures the amount of time elapsed from the time the watch is started to the time it is stopped. The following example shows the data structure set up in memory for a timer named TIMRNAME. The timer macro, CRTIMERS, is used to set up timer data areas.

Example:

```
             ALIGN 4
    TIMRNAME DCF   C'TIMRNAME'           TIMER NAME (8 CHAR MAX)
             DCF   0                     TIMER COUNTER
             DCF   0                     TIMER START VALUE
             DCF   0                     ACCUMULATED TIME
             DCF   0                     REGISTER SAVE AREA
```

The timer macros are used to set the watch and read the accumulated time after a specified interval has elapsed. The timer macros are listed in Table 6-1.

TABLE 6-1 TIMER MACROS

MACRO	FUNCTION
CRTIMERS (NAME1[,NAME2,...])	Creates a data area for each named timer.
STRTIME NAME(,REG)	Starts the named timer.
STOPTIME NAME(,REG)	Stops the named timer.
GETIME NAME,REG	Gets the total time accumulated by the named timer.
READTCNT NAME,REG	Gets the number of intervals that have been timed by this timer.
RESETIME NAME	Resets accumulated time counts.

The following example demonstrates how these macros can be used to time the execution of a program and its subroutine.

Example:

```
    * Create a data area for the timer                              *
    * for MAIN and the timer for SUB                                *
    *                                                               *
          CRTIMERS (MAIN,SUB)
    *                                                               *
    * Start timer for MAIN.                                         *
    *                                                               *
```

```
START    EQU   *                                                       *
         STRTIME MAIN
           .
           .
           .                                                           *
         BAL   R15,SUBPROG
           .
           .
           .
* Stop timer for MAIN                                                  *
         STOPTIME MAIN
* Get total time accumulated by MAIN                                   *
* Timer.  Load into REG 0                                              *
         GETIME MAIN,R0
           .
           .
           .
* Log MAIN program execution time.                                     *
           .
           .
           .
* Get total time accumulated by SUB                                    *
* timer.  Load into REG 3
         GETIME SUB,R3
* Get number of intervals timed by                                     *
* SUB timer.  Load into R0                                             *
         READTCNT SUB,R0                                               *
           .                                                           *
           .                                                           *
           .                                                           *
* Compute average subroutine execution                                 *
* time.                                                                *
         DR    R2,R0                                                   *
           .
           .                                                           *
           .
SUBPROG  EQU   *
*                                                                      *
* Start timer for SUB                                                  *
*                                                                      *
       STRTIME SUB                                                     *
           .
           .
*          .                                                           *
* Stop timer for SUB                                                   *
*                                                                      *
         STOPTIME SUB
         BR    R15
```

Detailed descriptions of the timer macros can be found in the OS/32 System Macro Library Reference Manual.

6.8 WHERE TO GO FOR MORE INFORMATION

This chapter is intended to demonstrate assembly language programming techniques used in designing system level control programs that take advantage of the Model 3200MPS System multiprocessing capabilites. However, all the programming facilities available for writing system level control programs are not shown. Table 6-2 summarizes additional facilities and lists the manuals in which they are described.

TABLE 6-2 ADDITIONAL INFORMATION SOURCES FOR MODEL 3200MPS SYSTEM PROGRAMMING

MANUAL	PROGRAMMING METHODS DESCRIBED
Model 3200MPS System Instruction Set Reference Manual	This manual describes the machine instructions specific to the Model 3200MPS System processor. It also gives a detailed discussion of the APU processor states.
FORTRAN VII User Guide	This manual describes the Perkin-Elmer FORTRAN VII run-time library (RTL) routines available for writing a FORTRAN system level control program that performs the functions described in this chapter.
OS/32 Operator Reference Manual	This manual describes the operator commands that can be used to write a command substitution system (CSS) command file to perform SVC 13 mapping and control functions.
OS/32 Supervisor Call (SVC) Reference Manual	This manual gives more details on how to use SVC 6, SVC 2 code 23, and assembly language programming SVCs.
OS/32 System Macro Library Reference Manual	This manual describes the timer and SVC 13 macros.

APPENDIX A
OS/32 SUPPORTED INPUT/OUTPUT (I/O) DEVICES

		ATTRIBUTES								
TYPE	DEVICE	RD	WRT	TS	BIN	WAT	RND	FLP	INT	HLT
Card equipment	400 CPM card reader	x			x	x				
	1000 CPM card reader	x			x	x				
	High speed card reader/punch	x	x		x	x				
	High speed card reader/punch with separate print option	x	x		x	x				
Teletype reader/ punch	TTY Model 33 *	x	x		x	x				x
	TTY Model 35 *	x	x		x	x				x
	Carousel 35 with paper tape reader 132-character line	x			x	x				x
Teletype keyboard printer	TTY Model 33	x	x			x			x	x
	TTY Model 35	x	x			x			x	x
	Perkin-Elmer Carousel * 15, 30, 35, 132-character line *	x	x			x			x	x
	Perkin-Elmer Carousel * 15, 30, 35, 80-character line *	x	x			x			x	x

TYPE	DEVICE	ATTRIBUTES								
		RD	WRT	TS	BIN	WAT	RND	FLP	INT	HLT
Teletype keyboard printer (Continued)	Perkin-Elmer Carousel 300 ^ $	x	x			x			x	x
	Perkin-Elmer Carousel 300 with electronic format controls	x	x			x			x	x
Paper tape equipment	Paper tape reader/ punch	x	x		x	x				x
Printers	Low speed line printer		x			x				
	Letter quality printer		x			x				
	Character printer		x			x				
	Medium speed line printer		x			x				
	High speed line printer		x			x				
	Thermal page printer		x			x				
	Remote printer		x			x				
Tape Cassette	Intertape	x	x		x	x		x		
Magnetic tape	800 bpi	x	x		x	x		x		
	1600 bpi	x	x		x	x		x		
	1600/800 bpi dual density	x	x		x	x		x		
	6250 bpi	x	x		x	x		x		
	6250 bpi halfword mode controller (STC)	x	x		x	x		x		
	6250 bpi TELEX mag tape	x	x		x	x		x		

TYPE	DEVICE	ATTRIBUTES								
		RD	WRT	TS	BIN	WAT	RND	FLP	INT	HLT
Magnetic tape (Continued)	9-track, 75 ips, 800 bpi	x	x		x	x		x		
	9-track, 45 ips, 800/1600 bpi	x	x		x	x		x		
	9-track, 45 ips, 800 bpi	x	x		x	x		x		
	9-track, 45 ips, 1600 bpi	x	x		x	x		x		
Disks	1.5Mb HPT disk	x	x	x	x	x	x			
	2.5Mb moving head disk, removable	x	x	x	x	x	x			
	2.5Mb moving head disk, fixed	x	x	x	x	x	x			
	10Mb moving head disk (5Mb fixed, removable)	x	x	x	x	x	x			
	40Mb moving head disk,removable	x	x	x	x	x	x			
	67Mb disk, fixed	x	x	x	x	x	x			
	67Mb disk, remov-able	x	x	x	x	x	x			
	160Mb disk, fixed	x	x	x	x	x	x			
	256Mb disk, removable	x	x	x	x	x	x			
	256Mb disk, fixed	x	x	x	x	x	x			
	675Mb disk, fixed	x	x	x	x	x	x			
	68.6Mb disk	x	x	x	x	x	x			
	MSM 300 disk sys-tem (256Mb format-ed)	x	x	x	x	x	x			
	MSM 80 disk system (67Mb formatted)	x	x	x	x	x	x			

TYPE	DEVICE	ATTRIBUTES								
		RD	WRT	TS	BIN	WAT	RND	FLP	INT	HLT
Disks (Continued)	MSM 80F disk system	x	x	x	x	x	x			
	MSM 80F/HPT disk system	x	x	x	x	x	x			
	Vanguard 1 cartridge disk system	x	x	x	x	x	x			
Video display units (VDU)	Nonediting VDU $ ^	x	x			x			x	x
	Graphic display terminals $ ^	x	x			x			x	x
	Model 1200 VDU $	x	x			x			x	x
	Model 1100 VDU $ ^	x	x			x			x	x
	Model 550 VDU $ *	x	x			x			x	x
	Model 1250 VDU $	x	x			x			x	x
	Model 1251 VDU $	x	x			x			x	x
Floppy disk	Floppy disk	x	x		x	x	x			
8-line interrupt module	8-line interrupt module					x				
Digital multiplexor	Digital multiplexor controller	x	x		x	x	x			
Conversion equipment	Real-time analog system with internal clock	x	x		x	x				
	Real-time analog system with user supplied external clock	x	x		x	x				

		ATTRIBUTES								
TYPE	DEVICE	RD	WRT	TS	BIN	WAT	RND	FLP	INT	HLT
Analog I/O controller	Analog input controller	x			x	x	x			
	Analog output controller		x		x	x	x			
Digital I/O controller	Digital I/O and analog output system	x	x		x	x	x			

LEGEND

*	CLI - Current loop interface
^	CLCM - Current loop communications multiplexor
$	RS232C

ATTRIBUTES

RD	READ
WRT	WRITE
TS	TEST & SET
BIN	BINARY
WAT	WAIT
RND	RANDOM
FLP	FILE POSITION
INT	INTERACTIVE
HLT	HALT I/O

APPENDIX B
SUPPORTED VERTICAL FORMS CONTROL (VFC) CHARACTER SET

HEX	CHAR	OPERATIONS AFFECTING LINE SPACING
09	HT	Horizontal tab
OB	VT	Set vertical tabs
		(EVFU, no print)
20	b	1 line b/print
2B	+	No line advance
2D	-	3 lines b/print
30	0	2 lines b/print
31	1	Top of form b/print
32	2	Select VFU-2 b/print
33	3	Select VFU-3 b/print
34	4	Select VFU-4 b/print
35	5	Select VFU-5 b/print
36	6	Select VFU-6 b/print
37	7	Select VFU-7 b/print
38	8	Select VFU-8 b/print
39	9	Select VFU-9 b/print
41	A	Select VFU-10 b/print
42	B	Select VFU-11 b/print
43	C	Select VFU-12 b/print
45	E	1 line a/print
46	F	No line advance
47	G	3 lines a/print
48	H	2 lines a/print
49	I	Top of form a/print
4A	J	Select VFU-2 a/print
4B	K	Select VFU-3 a/print
4C	L	Select VFU-4 a/print
4D	M	Select VFU-5 a/print
4E	N	Select VFU-6 a/print
4F	O	Select VFU-7 a/print
50	P	Select VFU-8 a/print
51	P	Select VFU-9 a/print
52	R	Select VFU-10 a/print
53	S	Select VFU-11 a/print
54	T	Select VFU-12 a/print
60	'	No line advance
61	a	1 line b/print
62	b	2 lines b/print
63	c	3 lines b/print
64	d	4 lines b/print
65	e	5 lines b/print
66	f	6 lines b/print
67	g	7 lines b/print

HEX	CHAR	OPERATIONS AFFECTING LINE SPACING
68	h	8 lines b/print
69	i	9 lines b/print
6A	j	10 lines b/print
6B	k	11 lines b/print
6C	l	12 lines b/print
6D	m	13 lines b/print
6E	n	14 lines b/print
6F	o	15 lines b/print
70	p	16 lines b/print
71	q	17 lines b/print
72	r	18 lines b/print
73	s	19 lines b/print
74	t	20 lines b/print
75	u	21 lines b/print
76	v	22 lines b/print
77	w	23 lines b/print
78	x	24 lines b/print
79	y	25 lines b/print
7A	z	26 lines b/print
7B	{	27 lines b/print
7C	!	28 lines b/print
7D	}	29 lines b/print
7E	~	30 lines b/print
7F	DEL	31 lines b/print
80		32 lines b/print
81		33 lines b/print
82		34 lines b/print
83		35 lines b/print
84		36 lines b/print
85		37 lines b/print
86		38 lines b/print
87		39 lines b/print
88		40 lines b/print
89		41 lines b/print
8A		42 lines b/print
8B		43 lines b/print
8C		44 lines b/print
8D		45 lines b/print
8E		46 lines b/print
8F		47 lines b/print
91		48 lines b/print
92		50 lines b/print
93		51 lines b/print
94		52 lines b/print
95		53 lines b/print
96		54 lines b/print
97		55 lines b/print
98		56 lines b/print
99		57 lines b/print
9A		58 lines b/print
9B		59 lines b/print

HEX	CHAR	OPERATIONS AFFECTING LINE SPACING
9C		60 lines b/print
9D		61 lines b/print
9E		62 lines b/print
9F		63 lines b/print
A0		64 lines b/print
A1		65 lines b/print
A2		66 lines b/print
A3		67 lines b/print
A4		68 lines b/print
A5		69 lines b/print
A6		70 lines b/print
A7		71 lines b/print
A8		72 lines b/print
A9		73 lines b/print
AA		74 lines b/print
AB		75 lines b/print
AC		76 lines b/print
AD		77 lines b/print
AE		78 lines b/print
AF		79 lines b/print
B0		No line space
B1		1 line a/print
B2		2 lines a/print
B3		3 lines a/print
B4		4 lines a/print
B5		5 lines a/print
B6		6 lines a/print
B7		7 lines a/print
B8		8 lines a/print
B9		9 lines a/print
BA		10 lines a/print
BB		11 lines a/print
BC		12 lines a/print
BD		13 lines a/print
BE		14 lines a/print
BF		15 lines a/print
C0		16 lines a/print
C1		17 lines a/print
C2		18 lines a/print
C3		19 lines a/print
C4		20 lines a/print
C5		21 lines a/print
C6		22 lines a/print
C7		23 lines a/print
C8		24 lines a/print
C9		25 lines a/print
CA		26 lines a/print
CB		27 lines a/print
CC		28 lines a/print
CD		29 lines a/print
CE		30 lines a/print
CF		31 lines a/print

HEX	CHAR	OPERATIONS AFFECTING LINE SPACING
D0		32 lines a/print
D1		33 lines a/print
D2		34 lines a/print
D2		35 lines a/print
D3		36 lines a/print
D4		36 lines a/print
D5		37 lines a/print
D6		38 lines a/print
D7		39 lines a/print
D8		40 lines a/print
D9		41 lines a/print
DA		42 lines a/print
DB		43 lines a/print
DC		44 lines a/print
DD		45 lines a/print
DE		46 lines a/print
DF		47 lines a/print
E0		48 lines a/print
E1		49 lines a/print
E2		50 lines a/print
E3		51 lines a/print
E4		52 lines a/print
E5		53 lines a/print
E6		54 lines a/print
E7		55 lines a/print
E8		56 lines a/print
E9		57 lines a/print
EA		58 lines a/print
EB		59 lines a/print
EC		60 lines a/print
ED		61 lines a/print
EE		62 lines a/print
EF		63 lines a/print
F0		64 lines a/print
F1		65 lines a/print
F2		66 lines a/print
F3		67 lines a/print
F4		68 lines a/print
F5		69 lines a/print
F6		70 lines a/print
F7		71 lines a/print
F8		72 lines a/print
F9		73 lines a/print
FA		74 lines a/print
FB		75 lines a/print
FC		76 lines a/print
FD		77 lines a/print
FE		78 lines a/print
FF		79 lines a/print

INDEX

A

Account reporting utility 1-9
Accounting transaction file 1-9
Analog input controller
   supported record lengths 5-14
Analog output controller 5-15
Analog to digital converter 5-13
APU. See auxiliary processing unit.
Assigning a task
   LPU mapping 6-2
ATF. See accounting transaction file.
Auxiliary processing unit 1-2, 3-31, 3-32
   active task 3-40
   commands 3-48, 3-50
   configuration options 3-35, 3-39
   control options 3-45, 3-47, 3-48
   control rights 3-35, 3-40, 3-44
   error codes 3-53, 3-54
   exclusive task 3-41
   hardware status 3-36, 3-46, 3-50
   initialization 3-57
   mapping 3-41
   mapping rights 3-35, 3-40, 3-41
   mark-off 3-60
   mark-on 3-57
   monitor task 6-6
   number of LPUs 3-35
   operating states 6-4
   power-up link check 3-44
   ready queue 1-8, 1-9, 3-34, 3-44
   ready task 3-41
   scheduling tasks on 3-58
   software status 3-38, 3-39
   start-up 3-57
   status 3-34
   task execution 6-4
   task scheduling 6-4
   waiting task 3-40

B

Bare disk I/O 3-20
Bare disk privileges 2-6, 3-20, 3-23, 5-11, 5-12
Basic data communications subsystem
   protocols 1-14
BIOC driver
   horizontal tabbing 5-2
Buffer access 6-13

C

Card reader/punch devices
   ASCII operations 5-4, 5-5
   binary operations 5-4, 5-5
   Hollerith code translation 5-5, 5-6
   image operations 5-5, 5-6
   supported record lengths 5-4
Card readers
   ASCII operations 5-3
   binary operations 5-3
   Hollerith code translation 5-4
   image operations 5-4
   supported record lengths 5-3
Central processing unit 1-2
   ready queue 1-5, 1-8, 1-9
   receive queue 1-8, 1-9
   roll-in queue 1-5, 1-8, 1-9
Circular list 4-4, 4-6, 4-7
Command processor subsystem
   priority level of 1-16
Command substitution system 1-16, 3-6
Console driver 1-15
Console monitor subsystem 1-15
Contiguous files 5-11, 5-12
CPU. See central processing unit.
CSS. See command substitution system.

D

D-tasks 2-1, 2-6, 3-20, 3-28, 5-11, 5-12
Data collection facility 1-9
Data management system 1-2
Data structures 2-3
Designing tasks
   simultaneous processing 6-1
Device dependent I/O 1-14
Device drivers 1-17
Device independent I/O 1-14
Diagnostic tasks. See d-tasks.
Digital I/O controller
   binary operations 5-15, 5-16
Digital multiplexor 5-13
Disk devices
   random reads 5-11
   random writes 5-11
   retry on error 5-11
   sequential read 5-11
   sequential writes 5-11
   supported record lengths 5-11
Disk directory 3-28
Dispatch priority 1-5
DMS/32. See data management system.
Dynamic scheduling 1-5
Dynamic time-slice 1-6, 1-7

E

E-tasks 2-1, 3-20, 3-23, 3-25, 3-26, 3-28, 5-11
   characteristics of 2-2
   data structures 2-3
   programming of 2-2, 2-3
Error code byte 3-50, 3-53, 3-54
Error log hardware 1-13
Error recording subsystem 1-13
Error reporting utility 1-13
Executive tasks. See e-tasks.
Extendable contiguous files 5-11, 5-12
Extended close function 3-28

F

File account privileges 2-6
File handlers 5-11, 5-12
File management subsystem 1-12
Files 1-12
Floating point subsystem 1-17
Floppy disk
   random reads 5-12
   random writes 5-12
   sequential reads 5-12
   sequential writes 5-12
   supported record lengths 5-12

G

Gapless I/O 5-11
Global task common 1-10
GSIG instruction 6-11

H

Hardware-echo 5-2
Horizontal tabbing 5-1, 5-2

I

I/O operations 1-13
I/O subsystem 1-12, 1-15
ICONT 4-27
   use of 4-13
ICREATE 4-16
   use of 4-8
IERRTST 4-34
   use of 4-14
IGET 4-24
   use of 4-12
Impure segment 1-14
Indexed files 5-11, 5-12
Input/output. See I/O.
Instrument Society of America 5-15
Integrated transaction controller 1-2
Intercept macros 4-1
   error codes 4-15
   ICONT 4-25
   ICREATE 4-16
   IERRTST 4-34
   IGET 4-24
   IPROCEED 4-28
   IPUT 4-26
   IREMOVE 4-22
   IROLL 4-29
   ITERM 4-30
   ITRAP 4-32
Intercept path 4-4, 4-8, 4-9
   end of task operation 4-13

Intercept path (Continued)
  example 4-13
  full control 4-11, 4-12
  identifier 4-8
  monitor control 4-12
  programming guidelines 4-13
  removing 4-14
Intercepting task 4-11, 4-12, 4-13
  use with MTM 4-2
Internal interrupt subsystem 1-16
Internal reader 3-6, 3-10
  command buffers 3-11
Interval timer queue 1-11
IPROCEED 4-28
  use of 4-12
IPUT 4-25
  use of 4-12
IREMOVE 4-22
  use of 4-14
IROLL 4-29
  use of 4-11
ISA. See Instrument Society of America.
ITC. See integrated transaction controller.
ITERM 4-30
  use of 4-13
ITRAP 4-32

J,K

Job accounting subsystem 1-9

L

LFC. See line frequency clock.
LIB. See loader information block.
Line frequency clock 1-11
Line printer driver
  horizontal tabbing 5-2
Line printers
  ASCII operations 5-9
  byte limit of transfer 5-10
  erasing of 5-10
  gapless I/O 5-11
  image operations 5-9
  read operations 5-10
  recording density 5-10
  retry on error 5-10
  standard I/O 5-10
  supported record lengths 5-9
  write operations 5-10
Loader and segmentation subsystem 1-13
Loader information block 1-13
Local memory 1-10
Logical processing unit 1-2, 1-8, 3-32
Logical processor mapping table 1-2, 3-32, 3-41, 3-59
LPMT. See logical processor mapping table.
LPU number 3-57
LPU. See logical processing unit.

M

Magnetic tape 5-9
Maximum priority 1-5
Memory
  types of 1-10
Memory conflicts
  buffer access 6-13
  system deadlock 6-13
  test and set 6-13
Memory diagnostics subsystem 1-13
Memory management subsystem 1-10
Memory map 1-13
Model 3200MPS System 1-2, 1-11, 1-17, 3-31, 3-34
  assigning a task 6-2
  designing tasks for 6-1
  memory conflicts 6-13
  performance advantage 1-2
  real-time performance 6-14
  task rescheduling 6-6
  task scheduling 1-8, 6-4
MTM. See multi-terminal monitor.
Multi-terminal monitor 1-2

N

Non-privileged instructions 2-6
Non-privileged u-tasks 2-5
Nonbuffered indexed files 5-11, 5-12

O

OS/32
  account reporting 1-9
  basic data communications 1-14
  command processor 1-15
  console monitor 1-15
  data collection facility 1-9
  data structures 2-3
  error recording 1-13
  file management 1-12
  I/O operations 1-12

OS/32 (Continued)
   interrupt servicing 1-16
   job accounting 1-9
   linkage editor 1-1
   loader and segmentation 1-13
   memory diagnostics 1-13
   memory management 1-10
   multiprocessing support 1-2
   optional supervisor call 1-17
   roll function 1-1
   software support summary 1-4
   subsystems 1-3
   supervisor calls 3-1
   supported I/O devices 5-1, A-1
   system initialization 1-16
   task management 1-5
   timer management 1-10
   virtual task manager 1-1

P,Q

Paper tape equipment
   ASCII operations 5-8
   binary operations 5-8
   image operations 5-8
   supported record lengths 5-8
PIC. See precision interval clock.
Power-up link check 3-44
Precision interval clock 1-11
Private segment table 3-14
Privileged instructions 2-1, 2-6
Privileged operating state 2-1
Privileged tasks 2-1, 2-1
   extended SVC 7 function codes 3-20
   types of 2-1
Privileged u-tasks 2-1, 2-6, 3-20, 3-28
Program status word 1-7, 3-50
Pseudo device
   use of generic naming 4-9
Pseudo task
   use of generic naming 4-9
PST. See private segment table.
Pure segment 1-14

R

RDB. See request descriptor block.
Ready queue 1-6, 1-8, 1-9
Real-time performance
   timer macros 6-14
Receive queue 1-8, 1-9
Reliance 1-2
Relocation/protection hardware 1-14, 2-6, 2-6
   programming of 2-2
Request descriptor block 4-36
   buffers 4-4, 4-6, 4-14
   fields 4-5
Response byte 3-50
   bit definitions 3-51, 3-52
Roll function 1-1
Roll-in queue 1-6, 1-8, 1-9
RSCH instruction 6-11
RTSM 1-11
Run priority 1-5, 1-7

S

SCL. See segment control list.
Segment control list 1-10
Shared memory 1-10
Software density selection 1-12
SPT. See system pointer table.
Standard I/O
   parity error 5-10
   record gap 5-10
   supported record lengths 5-10
Supervisor call interception
   caller mode 4-8
   creating intercept paths 4-8
   error handling 4-14
   functional summary 4-10
   macros 4-1
   operation of 4-3, 4-4
   recipient existent mode 4-8, 4-10
   recipient nonexistent mode 4-8, 4-9, 4-10
   rollable tasks 4-11
   sample programs 4-36, 4-38
   task preparation for 4-4
   termination of 4-13, 4-14
Supervisor calls. See SVC.
SVC 3-1
SVC 0 3-3
SVC 1 6-9
   interception 4-2, 4-10

SVC 13
  APU rescheduling 6-6
  control sequence 6-10
  error status codes 3-54, 3-55, 3-56
  example of 6-5, 6-10
  function codes 3-31
  mapping functions 6-4
  task scheduling 6-4
  use of 3-57, 3-58, 3-59
SVC 13 code 0
  coding 3-32
  data buffer format 3-34
  parameter block 3-32
SVC 13 code 1 3-34, 3-50
  coding 3-35
  data buffer format 3-37
  parameter block 3-35
SVC 13 code 2 3-41
  coding 3-42
  option coding sequences 3-57
  options 3-42
  parameter block 3-42
SVC 13 code 3 3-44, 3-50, 6-9, 6-11
  coding 3-45
  control options 3-45, 3-47, 3-48
  option coding sequences 3-57
  parameter block 3-45
SVC 14 1-17
SVC 2 code 0
  coding 3-4
  functional details 3-5
  parameter block 3-4
SVC 2 code 14
  coding for option 0 3-7
  coding for option 1 3-9
  example 3-11
  parameter block for option 0 3-6
  parameter block for option 1 3-8
  programming considerations 3-10
  status codes 3-10
SVC 2 code 23 6-9, 6-11
SVC 2 code 26
  coding 3-12
  condition codes 3-13
  example 3-13
  parameter block 3-12
SVC 2 code 27
  coding 3-14
  condition codes 3-16
  option codes 3-15
  parameter block 3-14
SVC 2 code 7 6-9
  interception 4-2, 4-11
SVC 3
  interception 4-2, 4-11
SVC 6
  example of 6-3, 6-12
  interception 4-2, 4-11
  loading a task 6-2
  suspend task 6-12
SVC 6 system task release 3-17
  coding 3-18
  parameter block 3-18
SVC 7
  interception 4-2, 4-11
SVC 7 bare disk assignment 3-20
  coding 3-21
  parameter block 3-21
SVC 7 code 0
  coding 3-23
  parameter block 3-23
SVC 7 code X'FF80' 3-28
  coding 3-29
  parameter block 3-29
SVC 7 device rename
  coding 3-25
  parameter block 3-25
SVC 7 device reprotect 3-26
  coding 3-27
  parameter block 3-27
System deadlock
  avoiding 6-13
System initialization subsystem 1-16
System journal 3-4
System memory 1-10
System pointer table 2-2
System task 3-17
System time-slice 1-6

T

Tape cassette
  ASCII operations 5-9
  binary operations 5-9
  continuous mode operations 5-9
  image operations 5-9
  supported record lengths 5-9
Task
  dynamic scheduling 1-5
  execution state 1-7
  impure segment 1-14
  priority levels 1-5
  pure segment 1-14
  user execution state 1-8
Task common 1-14
Task control block 1-5, 1-8
Task event trap 4-4
  handler 4-8

Task event trap (Continued)
   register saving 4-8
Task image 1-13, 1-14
Task management 1-5
Task priority 1-5
Task priority levels
   dispatch 1-5
   maximum 1-5
   run 1-5, 1-7
   task 1-5
Task queue trap 4-12
Task rescheduling
   connecting the APU, example of 6-7
   connecting the APU, parameters 6-8
   GSIG instruction 6-11
   monitoring and preempting 6-6
   RSCH instruction 6-11
   task queue, example of 6-7
   transferring a task 6-12
   trap handling 6-6
   verifying task transfer 6-11
Task segment table 1-14
Task status word 4-8
   CPU override status bit 1-8
Task transfer
   example of 6-12
   from APU to CPU 6-12
Task traps 1-7, 1-13, 1-17
TCB. See task control block.
Teletype keyboard/printer
   ASCII operations 5-7
   image operations 5-8
   supported record lengths 5-7
Teletype reader/punch
   ASCII operations 5-6, 5-7
   binary operations 5-6, 5-7
   image operations 5-7
   supported record lengths 5-6
Test and set 6-13
   example of 6-14
Time of day queue 1-11
Time-slicing
   types of 1-6
Timer macros
   CRTIMERS 6-14
   example of 6-15
   GETIME 6-14
Timer macros (Continued)
   READTCNT 6-14
   RESETIME 6-14
   STRTIME 6-14
Timer management subsystem 1-11
Timer queues 1-11
TMQ. See timer queues
TSW. See task status word.
TTY. See teletype.

U

U-tasks 2-1, 5-11, 5-12
   non-privileged 2-5
   priviledged 2-5
Universal clock 1-11
User supervisor call subsystem 1-17
User tasks. See u-tasks.

V

VDU. See video display unit.
Verifying task transfer to an APU
   example of 6-11
Vertical forms control
   functions of 5-1
   supported characters 5-1, B-1
   theory of operating 5-2
VFC. See vertical forms control.
Video display unit
   ASCII operations 5-12
   image operations 5-13
   supported record lengths 5-12
Virtual task manager 1-1
Volume mnemonic table 3-12
VTM. See virtual task manager.

W-Z

WCS. See writable control store.
Writable control store 3-35

8-line interrupt module 5-13

CUT ALONG LINE

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

Fold and Staple

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

6434

STAPLE STAPLE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

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

POSTAGE WILL BE PAID BY ADDRESSEE

PERKIN-ELMER

Computer Systems Division
2 Crescent Place
Oceanport, NJ 07757

TECH PUBLICATIONS DEPT. MS 322A

FOLD FOLD

STAPLE STAPLE

PERKIN-ELMER
Technical Systems Division

D O C U M E N T A T I O N   C H A N G E   N O T I C E

The purpose of this documentation change notice (DCN) is to provide a quick and efficient way of making technical changes to technical manuals before they are formally updated or revised.

The manual affected by these changes is:

<u>48-040 F00 R02</u> <u>OS/32 System Level Programmer Reference Manual</u>

- Page 3-38

  Under APU software status, replace the statement, "status after the last power fail", to "previous status".

- Page 3-38

  Add the following paragraph to APU software status:

  > The "previous status" would be returned in the caller's buffer when the following occurs:
  >
  > - Global power failure
  > - Local APU power failure
  > - APU hardware failure
  > - RTSM link failure

- Page 3-38

  In Figure 3-16, replace "Last power failure status" with "previous status".

- Page 3-39

  In Table 3-4 replace "Last power failure status" with "previous status".