PERKIN-ELMER

OS/32
SYSTEM LEVEL

Programmer Reference Manual

48-040 F00 R03

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

Printed in the United States of America

TABLE OF CONTENTS

CHAPTERS (Continued)

CHAPTERS (Continued)

CHAPTERS (Continued)

APPENDIXES

FIGURES

FIGURES (Continued)

TABLES

PREFACE

This manual describes 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 subsystems it supports. Chapter 2 describes the privileged task types supported by OS/32. Chapter 3 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. Chapter 4 contains a functional description of the supervisor call (SVC) interception feature. The vertical forms control (VFC) feature is described in Chapter 5, along with other device-independent and device-dependent features supported by OS/32.

Revision 03 is intended for use with the OS/32 R07.2 software release or higher. It introduces a change in the method of task execution by the auxiliary processing units (APUs) within the Model 3200MPS System. Therefore, Chapter 3 has been totally reorganized. Material related to OS/32 SVCs is no longer included in this manual. This material is now documented in the OS/32 Supervisor Call (SVC) Reference Manual.

For 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 virtual task manager (VTM) allows the memory requirements of a task running under OS/32 to exceed available task memory.

The roll function allows segments 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 rolling of a task when necessary (e.g., when the task must be able to respond to real-time events).

VTM has a 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 the following 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 on-line 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 assignment, APU control, etc.). See the OS/32 Supervisor Call (SVC) Reference Manual for more information.

OS/32 defines a set of logical processing units (LPUs) that are used to schedule tasks to APU queues for execution on an APU. Tasks are assigned to an LPU that is mapped to an APU queue. The logical processor mapping table (LPMT) defined by OS/32 contains the mapping arrangement between the LPUs and APU execution queues. (More than one LPU can be mapped to an APU queue.)

Each APU in a Model 3200MPS System is assigned a unique identifying number and is assigned to an APU queue. (More than one APU can be assigned to a queue.) Tasks on the APU queue execute on an APU assigned to the queue.

If a task is mainly computation 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 3 for more information on programming within a Perkin-Elmer Model 3200MPS System environment.

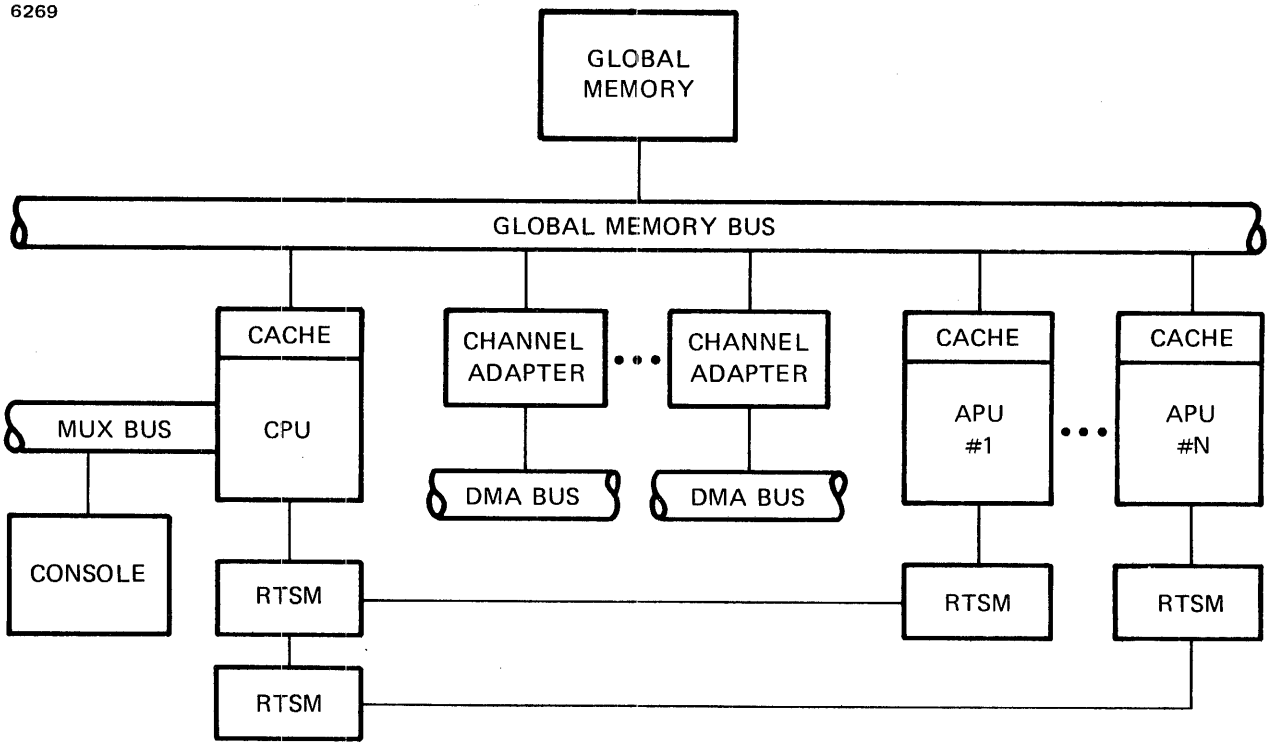


Figure 1-1 Typical Model 3200MPS System Configuration

1.2 SOFTWARE SUBSYSTEMS

OS/32 consists of the following subsystems:

- Task management
- Job accounting
- Memory management
- Timer management
- File management
- 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 14 (SVC14)
- Floating point

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

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

TYPE	SOFTWARE PRODUCT	STANDARD	OPTIONAL
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	
	ITC*		
	Writable control store (WCS)		x
	MTM		x
Program debugging	Automatic interactive debugging system (AIDS)	x	
	DEBUG/32		x
Data base management	DMS*		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
	Ethernet communications	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

TABLE 1-1 PERKIN-ELMER OS/32 SOFTWARE SUPPORT (Continued)

TYPE	SOFTWARE PRODUCT	STANDARD	OPTIONAL
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	x	
	Backup	x	
	Error Reporting	x	
	Disk Dump	x	
	Dump Print	x	
	Mirror Disk Synchronizaton		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 multitasking 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 to 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., the 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 can 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 one of the following occurs.

- It is paused by the system operator.
- It is cancelled by the system operator, user or 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 an SVC that places it in a wait, paused or dormant state.
- It initiates I/O to a device.
- 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 limits 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.

At system generation (sysgen), system time-slicing can be enabled through the use of the SLICE command. This allows time-slice scheduling to be activated automatically by the system. Thereafter, the operator SET SLICE command can be used to override the SLICE command.

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 slice is measured in units of line frequency clock (LFC) ticks (one LFC tick = 8.333ms).

Run priority is adjusted whenever a task uses up a time-slice, is removed from a wait state, or has its priority modified by the operator SET PRIORITY command. 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 does not use 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 intratask 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 SVC14 traps. Asynchronous events originate outside of a task and include the task queue traps (e.g., I/O and timer completion, SVC6 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
- CPU roll-in queue
- APU execution queue

An APU execution queue can be one of four different types:

- APU idle queue, not serviced by any processor
- APU private queue, serviced by a single APU
- APU shared queue, serviced by several APUs

- CPU/APU shared queue, serviced by the CPU whenever its own ready queue is empty and/or by one or more APUs; this is always APU execution queue 0.

Each APU execution queue can be designated either no-priority or priority-ordered. Priority-ordered queues can be either enforced or not enforced, depending upon whether a task put on the top of the queue preempts a currently executing task of lower priority. The CPU ready queue is priority-enforced, and the CPU receive queue is no-priority.

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 placed on one of the APU queues in the system. A task is transferred to an APU queue for processing only when all of the following conditions are true:

- The task must be executing in the user state, not in the system state.
- The task's "LPU-directed" status must be set. (In MTM, when the load-leveling executive (LLE) is active, subtasks of MTM cannot be LPU-directed unless the user has SVC6 privileges.)
- 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.)

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 queue. 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 queue.

Whenever it is not processing a task, the APU continually checks its APU 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 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 queue or to the CPU. In a no-priority APU queue, the task is placed at the bottom of the queue. In a priority APU queue, the task is placed behind all tasks of equal or higher priority, or at the queue top if there is no task of equal or higher priority on the queue. In a priority-enforced APU queue, the task is placed on the queue in the same manner as for a priority queue. In addition, whenever the task happens to be placed at the queue top, the operating system executes the preempt procedure to ensure execution of the highest priority tasks, even if a lower priority task is currently executing.

The task is returned to the CPU receive queue if it is rescheduled to the CPU, if a fault occurs, or if an SVC or operating system request occurs. 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 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 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.

Tasks running under MTM will run on APUs as determined by the LLE at a priority scheduled by the priority scheduling mechanism (PSM). When the LLE is active, MTM controls whether the task will be assigned to one of the APUs or to the CPU.

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 information gathered 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. This command also gives MTM users access to accounting data for a task monitored by MTM on their behalf.

NOTE

The OS/32 job accounting subsystem now reports 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 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 (UCLOCK).

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

- Time of day
- Device time-out
- Communications device time-out
- 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 SVC2 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 UCLOCK consists of an LFC and a precision interval clock (PIC). In a 60Hz system, the LFC generates an interrupt every 8.3ms or 120 times per second. In a 50Hz system, the LFC generates an interrupt every 10ms, or 100 times per second. The PIC interrupts when a task's requested time interval has expired or at intervals of 4,096ms, 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 a logical unit (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 bits per inch (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 processors. 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 read-out 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 and marked on and 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 irrecoverable 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 read-only 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, the zero-bit data link control (ZDLC) Channel Terminal Manager and the Ethernet Data Link Controller (EDLC), which support the synchronous data logic control (SDLC), high-level data link controller (HDLC) and advanced data communications control procedure (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 SVC6 to the command processor notifying it of a command to be processed. The command processor interprets the command and issues an SVC6 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 perform 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 SVC6 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 that 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 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 SVC6 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 SVC6 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 (SQS) 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. This subsystem does not support external I/O interrupts; they 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

SVC14 is provided as an optional SVC that can be defined by the user. On execution, the task receives a task trap for SVC14. See the OS/32 Application Level Programmer Reference Manual for information on how to implement the SVC14 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. 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 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)
- 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 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 two macro libraries that contain OS/32 and multi-terminal monitor (MTM) data structures. The OS/32 data structure macro library is stored in file SYSSTRUC.MLB. Table 2-1 contains a list of the macros and corresponding data structures in this library. Data structures specific to the MTM subsystem are stored in file MTMSTRUC.MLB. The contents of this library are listed in Table 2-2.

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 MACRO LIBRARY

MACRO	DATA STRUCTURE
$ACB	Directory access control block (ACB)
$AOPT	Auxiliary processing unit (APU) options
$APB	Auxiliary processor block (APB)
APB	$APB, $APRC, $APS, $AOPT
$APRC	Passback reason codes and equates
$APS	APB status codes and equates
$APST	APU status codes, error codes and equates
$ATF	Account transaction file (ATF)
$CCB	Channel control block (CCB)
$CTX	U-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	Supervisor call (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	Intercept control block
$IVT	Initial value table

TABLE 2-1 OS/32 DATA STRUCTURES MACRO LIBRARY (Continued)

MACRO	DATA STRUCTURE
$LIB	Loader information block (LIB)
LIB	LIB and loader options
$LLE	Load leveling executive (LLE)
$LPMT	Logical processor mapping table (LPMT)
$LOPT	Loader options
$LSG	Load segment
$LTCB	Loader task control block (TCB) redefinitions
$MAGDCB	Magnetic tape DCB
$MERC	Memory error recording entry
$OCB	Overlay control block
$ODT	Overlay descriptor table (ODT) structure
$ORT	Overlay reference table
$PDCB	Primary (device-independent) DCB
$PFCB	Private FCB
$PSDCB	Pseudo DCB structure (device-dependent)
$PSTCB	Pseudo TCB
$PSW	Program status word (PSW)
$QH	SVC intercept queue handler structure
$QPB	Queue parameter block (QPB)
QPB	$QPB, $QPSTAT
$QPSTAT	QPB status
$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	SVC1 extended options masks
$SDCB	Pseudo print DCB structure
$SD	Send data message block
$SDE	Segment descriptor element
$SOPT	System options
$SPLMSG	Spooler message structures
$SPT	SPT
$SPTE	SPT extern definitions
$SPOL	Spooler message
$STE	Segment table entries (STEs)
$SPR	Segment privilege flags
$SVC1	SVC1
$SVC1$	SVC1 and SVC1 error codes
$SVC1ERR	SVC1 error codes
$SVC4	System SVC - reserved
$SVC5	SVC5 parameter block
$SVC6	SVC6 parameter block
$SVC7	SVC7 parameter block

TABLE 2-1 OS/32 DATA STRUCTURES MACRO LIBRARY (Continued)

MACRO	DATA STRUCTURE
$SVC7EXT	Extended SVC7 functions
$SVC7SPL	Spooler SVC7 parameter block
$SVC13	SVC13 parameter block
$SVC13$	$SVC13, $APST
$SVT	System value tab
$SYP	System space structure
$$SPT	SPT table definitions
$TABL$	Structure/extern generating macro
$TCB	TCB, $SDE, IOB, $TCB, $CTX
TCB	$TCB, $TOPT, $TSTT, $TWT, $TLFL, $PSTCB, $OCB, $TQE, $TFL, $TPRC, TQH
$TFL	TCB flags
$TKQ	Task queue head
$TLFL	Logical unit (lu) table of flags
$TMQ	Timer queue entry
$TOPT	Task options flags
$TPRC	Task passback codes
$TQE	Task event queue entry
$TQH	Task event queue header
$TQ27	SVC2 code 27 parameter block
$TSTT	Internal task status flags
$TSW	Task status word (TSW)
$TTB	APU trap block
$TWT	Task wait status flags
$UDL	User-dedicated locations (UDLs)
UDL	UDL and TSW
$UREGS	16 general user registers (U1 = register 1)
$VD	Volume descriptor
$VFCHARS	Vertical forms control (VFC) characters
$VFDCB	Common VFC DCB structure
$WAP	Read/write access matrix header structure

TABLE 2-2 MTM DATA STRUCTURES MACRO LIBRARY

MACRO	DATA STRUCTURE
$TERMUSR	Terminal user block
$AUF	Authorized user file (AUF) record
$MTMSTE	Terminal state definitions
$PRIV	User privileges
$VAR	Command substitution system (CSS) variable flags and structures
$BTQ	Batch queue header and entry structures
$CMB	Command buffer structure
$LMB	Log/broadcast message buffer structures
$CBH	Common buffer header structure
$CSTK	CSS pointer stack 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 fd. Bare disk privileges allow tasks to perform I/O operations to a bare disk device. See the OS/32 Supervisor Call (SVC) Reference Manual.

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 writable control store (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
PROGRAMMING IN A MODEL 3200MPS SYSTEM MULTIPROCESSING ENVIRONMENT

3.1 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 be executed on auxiliary processing units (APUs)
- The preparation of the APUs for task execution
- The assignment of tasks to the processors
- The establishment and control over the order of task execution
- The prevention of invalid data variables, caused when two tasks running on different processors concurrently read and modify a common data structure
- The measurement of real-time performance of the individual tasks in the system
- The customization of APU fault handling

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. For additional information on Model 3200MPS System programming, see Table 3-3 at the end of this chapter.

3.2 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.

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 is 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 an APU for execution. It should be remembered that execution of a computation-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 transfer back to the CPU for operating system support. All I/O-intensive tasks should be assigned to the CPU for execution.

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

OS/32 supports a multiprocessing configuration consisting of one CPU and one to nine APUs. The operating system schedules tasks for execution by arranging them in queues. These queues consist of a CPU ready queue and APU execution queues.

3.3.1 Queue Assignments

The CPU ready queue is intended for supervisor call (SVC) I/O-intensive tasks and is serviced by the CPU. The APU queues are numbered 0 through n where n represents the number of APUs in the system. They are intended for the computation-intensive tasks and are serviced by APUs assigned to them. APU queue 0 is serviced by the CPU when the CPU ready queue is empty. The APU execution queues numbered 1 and above are also referred to as APU-only queues.

The APU-only queues may have the following assignment possibilities:

- The queue is idle with no APUs assigned
- The queue is private and has one APU assigned
- The queue is shared with two or more APUs assigned

When the operating system is loaded, each of the APU-only queues is designated as a private queue and is assigned to one APU. The number of the queue will correspond to the number of the APU to which it is assigned. Subsequently, the APUs may be reassigned using a corresponding SVC13 control function or the operator command APC. To employ an SVC13 control function, a task must be linked using the OPTION APCONTROL command of LINK.

3.3.2 Auxiliary Processing Unit (APU) Operating States

OS/32 maintains two 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.
ENABLED	APU has successfully passed the power-up link check procedure and is ready for task execution.

All APUs are put into the DISABLED state upon operating system load or power restore. On a power fail restart, an attempt is made to upgrade each APU not disabled prior to the power fail to the ENABLED state.

The transition from one APU state to another can be accomplished along the paths shown in Figure 3-1. These transitions are executed by the corresponding SVC13 control functions or the operator APC command. The APU firmware logic requires resetting the APU state after it is disabled in order to be enabled again. The resetting is done using the appropriate button on the APU board or by powering down the APU cabinet.

9231

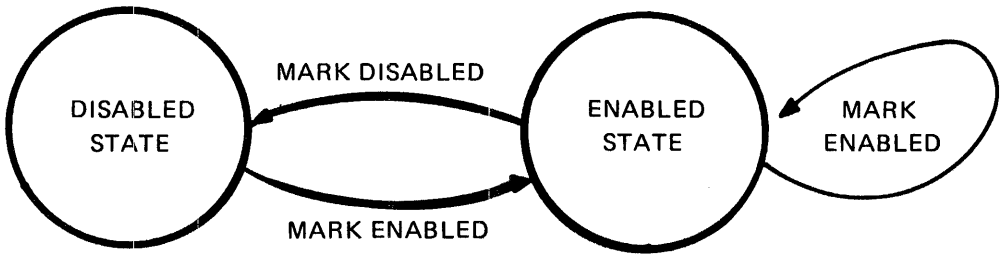


Figure 3-1 Valid APU Operating States

3.3.3 APU-Only Queue Operating States

OS/32 maintains three operating states for each APU-only queue, each differing in the degree of queue availability for task scheduling. These states are:

OFF	APU queue is not available for task scheduling.
ON EXCLUSIVE	APU queue has only a designated task scheduled to it. (Only an idle or private APU queue can be marked ON EXCLUSIVE.)
ON	APU queue is fully available for task scheduling.

All the APU-only queues are put into the OFF state upon operating system load. Upon a power fail restart, the load power fail monitor (LPFM) restores the queue states. The queue 0 is always maintained in the ON state.

The transition from one APU queue state to another can be accomplished along the paths shown in Figure 3-2. These transitions are executed by the corresponding SVC13 mapping functions or the operator QUEUE command. To use an SVC13 mapping function, a task must be linked using an OPTION APMAPPING command of LINK.

6275-1

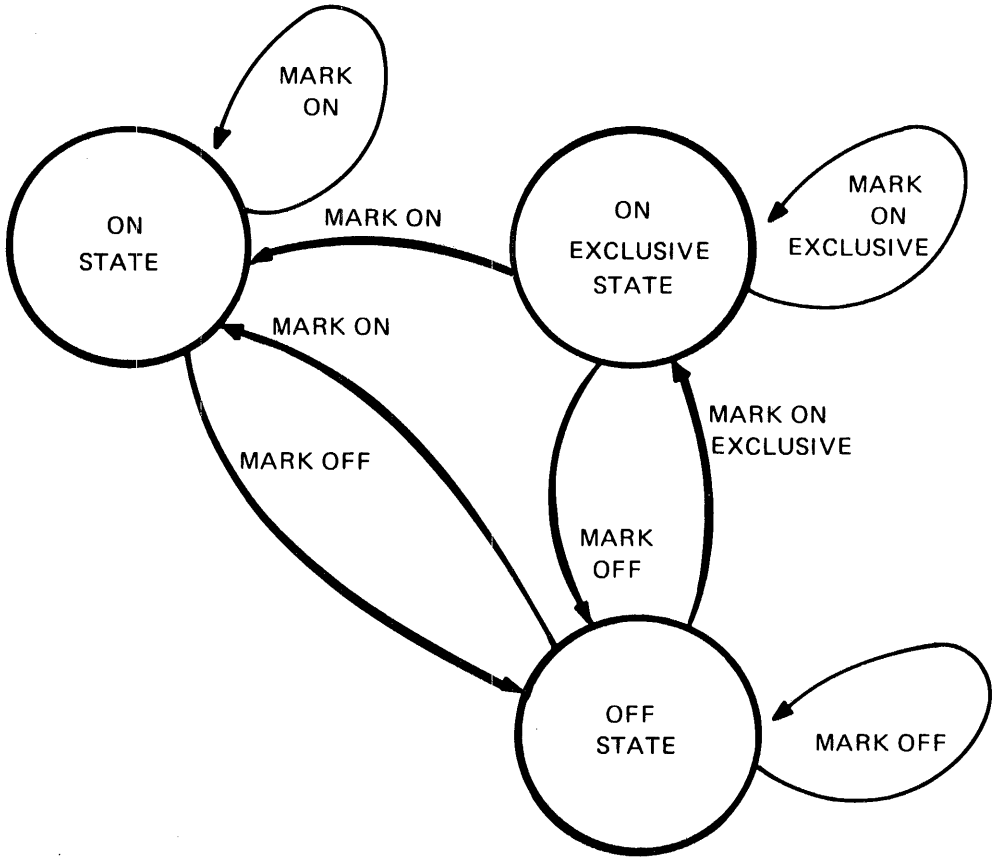


Figure 3-2 Valid APU Queue Operating States

```
*****************************************************************
*       The following code demonstrates how SVC13               *
*       is used to enable an APU and mark on the queue          *
*       for task scheduling. This example does not              *
*       check for SVC13 execution errors. A task incor-         *
*       porating this code must be linked using a LINK          *
*       command OPTION APCONTROL, APMAPPING.                    *
*****************************************************************
```

Example:

```
            $SVC13
            ALIGN 4
   ENABLE   DS    SVC13    ALLOCATE STORAGE FOR SVC 13 PARBLK
   ENABLEE  EQU   *
   *GAIN CONTROL RIGHTS, ENABLE APU, START APU, ASSIGN TO QUEUE,
   *RELEASE CONTROL RIGHTS
            ORG   ENABLE+SV13.OPT
            DB    X'CD'
            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   ENABLE+SV13.USE
            DCX   3                    ASSIGN APU TO QUEUE
            ORG   ENABLEE
   ***BUILD SVC 13 PARAMETER BLOCK FOR MARKING QUEUE
            ALIGN 4
   MARK     DS    SVC13.               ALLOCATE STORAGE FOR SVC 13 PARBLK
   MARKE    EQU   *
   *GAIN MAPPING RIGHTS, MARK QUEUE ON, MAP LPU,
   *RELEASE MAPPING RIGHTS
            ORG   MARK+SV13.OPT
            DB    X'B1'
            ORG   MARK+SV13.FUN
            DB    2                    FUNCTION CODE=2
            ORG   MARK+SV13.DOP
            DB    2                    LPU NUMBER TO BE MAPPED
            ORG   MARK+SV13.APN
            DB    3                    QUEUE TO MAP LPU TO
            ORG   MARKE
   ******ISSUE SVC 13 TO ENABLE APU AND MARK QUEUE*******
            SVC   13,ENABLE            ENABLE APU
            SVC   13,MARK              MARK QUEUE ON
```

3.3.4 Logical Processing Unit (LPU) Mapping

For the purpose of directing tasks to the queues, OS/32 defines LPUs ranging from 0 to 255. LPUs are mapped into the APU queues while each task is associated with a particular LPU.

All the LPUs are initially mapped to queue 0 at operating system load time. LPUs 1 through 255 can later be mapped to other queues using a corresponding SVC13 mapping function or the operator LPU command. LPU 0 always remains mapped to queue 0.

3.4 ASSIGNING TASKS TO A PROCESSOR QUEUE

As mentioned above, each task in the Model 3200MPS System is associated with an LPU. The initial LPU value is established at task link editing time to be either LPU=0 by default or a value specified in the OPTION LPU command of Link. The LPU value may be changed at task load time or whenever the task is paused via a corresponding SVC6 function or with an operator OPTION LPU command.

Each task's LPU mapping (association with an APU queue) is enabled or disabled in a task status either by default or via a corresponding SVC6 function or via operator OPTION LPU and OPTION NLPU commands. By default, LPU mapping is disabled if the task is linked with LPU=0 and is enabled if the task is linked with a non-zero LPU. The operator OPTION LPU command, however, sets the specified LPU and also enables mapping even if LPU=0 was previously specified. The operator OPTION NLPU command enables mapping without changing the LPU number.

All tasks with mapping enabled are called LPU-directed tasks. OS/32 places the LPU-directed tasks onto corresponding APU queues. Tasks with mapping disabled and CPU-directed tasks are placed onto the CPU ready queue.

```
*****************************************************************
*                                                               *
*       This example loads and starts a copy of a task          *
*       and makes it LPU-directed via the SVC6 function.        *
*                                                               *
*****************************************************************


         $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!SFUN.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,C'APU1'
         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 SVC6 in the above example is executed, the task will be loaded into memory from the file (DIRECTED.TSK) with a workspace of 64 (X'40') bytes. When the task is started, the task manager dispatches it to the APU queue into which LPU2 is mapped.

3.5 CONTROLLING TASK ORDER OF EXECUTION

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, there is a choice of possibilities to control the order of task execution as described in this section.

3.5.1 Changing Auxiliary Processing Unit (APU) Task Queue Ordering

Each of the APU queues can be set to handle its assigned tasks through the following priority disciplines.

- The no-priority queue services tasks in a first-in/first-out (FIFO) order, regardless of task priority.
- The priority queue services its highest priority tasks first and its equal priority tasks in a FIFO order. No preemption of currently executing tasks by higher priority tasks will occur.
- The priority-enforced queue services its tasks in the same manner as the priority queue; however, higher priority tasks are allowed to preempt lower priority tasks being executed on the processor assigned to the queue.

At operating system load time, the queues are initially set with the following priority assignments or disciplines. See Table 3-1.

TABLE 3-1 QUEUE PRIORITY ASSIGNMENTS

QUEUE	PRIORITY DISCIPLINE
CPU Ready Queue	Priority-enforced
APU Queue 0	Priority-enforced
APU Queue 1 to n	No-priority

These initial settings can be subsequently altered via a corresponding SVC13 mapping function or with an operator QUEUE command.

```
**************************************************************
*       The following example uses SVC13 to change a        *
*       queue priority discipline.  If the discipline       *
*       of this queue prior to the SVC13 call was no-       *
*       priority, OS/32 will reorder the queue according    *
*       to the task priorities subsequent to the SVC13      *
*       call.  A task incorporating this code must be       *
*       linked using the LINK command OPTION APMAPPING.     *
**************************************************************
```

Example:

```
            $SVC13
            ALIGN 4
    DISCIP  DS    SVC13    ALLOCATE STORAGE FOR SVC13 PARBLK
    SETD    EQU   *
    *GAIN MAPPING RIGHTS, SET QUEUE DISCIPLINE,
    *RELEASE MAPPING RIGHTS
            ORG   DISCIP+SVC13.OPT
            DB    X'85'
            ORG   DISCIP+SVC13.FUN
            DB    2                    FUNCTION CODE=2
            ORG   DISCIP+SVC13.APN
            DB    3                    QUEUE NUMBER
            ORG   DISCIP+SVC13.USE
            DC    H'1'                 PRIORITY DISCIPLINE
            ORG   SETD
    *ISSUE SVC13 TO CHANGE QUEUE DISCIPLINE
            SVC13, DISCIP
```

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

The Model 3200MPS System provides facilities to monitor APU operation via the mechanism of task trap service. The APU reports its significant events to OS/32 by issuing asynchronous status signals.

An APU status signal is a byte with the following format:

PAR	RUN	NON-TASK	WAIT	RESP	ERROR	MOD1	MOD2
Bits: 0	1	2	3	4	5	6	7

PAR — is the parity bit which is adjusted to maintain odd parity for the byte.

RUN — is set to 1 if the APU is currently not idle. The APU may be executing task instructions or performing servicing functions (nontask) such as: selecting a task from the queue, releasing a task back to the queue, processing a task fault, etc.

NON-TASK — is set to 1 if the APU is performing any of the servicing functions.

WAIT — is set to 1 if the APU is idle and in the wait state imposed upon it by the last executed task.

RESP — is set to 1 if the APU is returning the status in response to a command from the CPU (normally an SVC13 read APU status function); this bit is always reset in an APU asynchronous status signal.

ERROR — is set to 1 if the APU has detected an error in system data structures and subsequently entered an idle state.

MOD1, MOD2 — are set or reset to supply some additional status information.

When the last three bits are set to 1, they have the values of 4, 2 and 1, respectively. The actual value of these three bits reflects the APU status condition as follows:

0	Undefined.
1	The APU has detected that the queue to which it is assigned is empty (contains no tasks).
2	The APU has rescheduled (released) a task to the APU queue as a result of an RSCH1 instruction in the task or as a result of an appropriate command issued via an SVC13 control function.
3	The APU has rescheduled a task to the CPU ready queue as a result of an RSCH 0 instruction in the task, an appropriate command issued via an SVC13 control function, a nonexecutable APU instruction (normally an SVC), or a task fault.
4	The APU has detected an error in data structures at an arbitrary moment.
5	The APU has detected an error in data structures while attempting to select a task from its queue.
6	The APU has detected an error in data structures while attempting to lock its queue. Each processor locks its assigned queue prior to manipulating it.
7	Undefined.

Detailed information regarding the data structure errors detected by the APU can be obtained using SVC13 read APU status function.

NOTE

In the absence of an APU monitor task, OS/32 reports APU errors via the operator console.

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 SVC6) 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 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                                *
      * read APU assignment information to obtain the                 *
      * number of APUs in the system.                                 *
      *                                                               *
      START    SVC   13,APUASGN
               LB    R1,BUFFER1+1          LOAD MAX APU NO.INTO R1
      * SET UP SVC 13 PARAMETER BLOCK TO                              *
      * FETCH APU STATUS                                              *
               LIS   R3,X'80'
               STB   R3,FETAPU+SV13.OPT    SET APU STATUS OPTION
               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   R3,40
               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 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.ROUTE
               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 Read APU Assignment Parameter Block and Buffer
          ALIGN 4
APUASGN   DS     SVC13
ENDPBK    EQU    *
          ORG    APUASGN+SV13.FUN
          DB     X'00'               SET FUNC CODE 0
          ORG    APUASGN+SV13.BUF
          DAC    BUFFER 1            DATA BUFFER ADDR
          ORG    APUASGN+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     40
*
*** 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 SVC1 or SVC2 code 7) or to a file designated for the APU output (via SVC1). In addition, this monitor program can be coded to run a certain task (TASK1) every ten minutes on a specific APU. To do this, the monitor sets an interval timer via SVC2 code 23. Upon expiration of the timer, the monitor task issues an SVC13 code 3 to preempt the current executing task on the APU, as shown below. This preemption mechanism is only allowed on no-priority queues. It is used when the overhead associated with maintaining a priority queue is to be avoided.

Example:

```
         SVC   13,PREQ
         .
         .
         .
         ALIGN 4
* PREEMPT TASK EXECUTION, RESTART APU
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 ØØØ'          TASK ID BUFFER
```

Execution of the above SVC13 will cause the monitor to gain control rights to the specified APU (APU1), 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 SVC13 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 queue.
- The APU's 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 ten minutes, the interval timer (via SVC2 code 23) should be reset so that the SVC13 code 3 to preempt the current APU task can be reissued when ten minutes have elapsed.

See the OS/32 Supervisor Call (SVC) Reference Manual for more information on SVC13, SVC6 and SVC2 code 23.

3.5.3 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 ready queue. The task to be transferred may be executing on an APU or waiting on its queue. The monitor task can transfer a task back to the CPU ready queue by issuing an SVC6, specifying the following function codes:

- Suspend (SFUN.SM)
- Transfer to CPU (SFUN.XCM)
- Release (SFUN.RM)

The suspend will transfer the task back to the CPU ready queue, and then the LPU-directed task status is reset. Upon release, the task will stay on the CPU ready queue and not be dispatched according to its LPU assignment.

Example:

```
            SVC   6,CPUDIR
                .
                .
                .
            ALIGN 4
            $SVC6
CPUDIR      DS    SVC6.
CPUDIRE     EQU   *
            ORG   CPUDIR+SVC6.ID
            DC    C'TASK1ØØØ'          ID OF TASK TO BE
*                                          TRANSFERRED
            ORG   CPUDIR+SVC6.FUN
* SET OTHER-DIRECTED, SUSPEND, TRANSFER TO CPU, & RELEASE FUNC CODES
* FOR TASK1
            DB    SFUN.DOM!SFUN.SM!SFUN.XCM!SFUN.RM
            ORG   CPUDIRE
```

Execution of this SVC6 causes TASK1 to be suspended (if it is not already in a wait state) and transferred to the CPU ready queue. Resetting the LPU-directed status directs the task manager to ignore its LPU mapping and to schedule this task for execution on the CPU ready queue. 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 SVC6 in the above example did not reset the LPU-directed status bit, the task will again be dispatched to the APU queue into which its LPU is mapped upon release from the suspended state.

3.5.4 Internal Task Control of Auxiliary Processing Unit (APU) Execution

A task can exercise control over its own execution on an APU through the SVC6 mechanism described above since SVC6 can be made self-directed; however, there are more efficient mechanisms achieving the same result that are particulary valuable for real-time and APU diagnostic applications.

1. A task wishing to relinquish use of an APU while remaining on the same processor queue may issue the following instruction:

```
RSCH R1,1
```

   The APU places the task at the queue tail and immediately picks up the task residing at the queue head. OS/32 will restore the queue order according to the queue discipline, if necessary.

2. A task wishing to transfer to the CPU indefinitely, may issue the following instruction:

```
RSCH R1,0
```

   The APU sends the task to the CPU and then immediately picks up the task residing at its queue head. OS/32 resets the tasks LPU-directed status, which prevents the task from going anywhere but the CPU receive queue.

3. A task wishing to transfer to an APU indefinitely, according to its LPU mapping, may issue the following instruction:

```
RSCH R1,2
```

   OS/32 insures that the task is scheduled to the appropriate APU queue according to the task's priority and the queue discipline.

4. A task may manipulate its TSW CPU-override status to enable or disable its transfer to the APU, for a given reason, to which the TSW corresponds. Bit TSW.CPOB (currently bit 8) prevents task scheduling to an APU queue when set to 1. This is necessary when a particular task fault, not a single instruction, should be executed on the APU.

3.5.5 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 queue. For example, suppose a task on the CPU is assigned to LPU3 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 queue into which LPU3 has been mapped. Suppose LPU3 is mapped to APU queue 3 and APU4 is assigned to this queue. To verify that the task is indeed executing on APU4, 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 that the task is currently executing. See the Model 3200MPS System Instruction Set Reference Manual for more information on the RSCH and GSIG instructions.

3.5.6 Customizing Auxiliary Processing Unit (APU) Fault and Supervisor Call (SVC) Handling

OS/32 allows customization of fault and SVC handling by the APUs. When consistently pursued, this route may allow reduction of the task traffic between the APUs and the CPU caused by SVCs, or it may provide for APU I/O handling, "invisible" to OS/32, and subsequently more efficient.

As an example of this customized handling, an APU can be made to wait for a task return while the task fault or SVC is processed by the CPU. This may be needed to leave private queue orders undisturbed by occasional SVCs.

This feature is not fully supported by OS/32 and, therefore, is intentionally made difficult to use. However, software tools may be easily developed to exploit the customization feature.

In order to allow for custom processing of the faults and SVCs in a given task by the APUs, the following actions must be performed:

- An APU trap block has to be allocated in memory. This block will contain pairs of fullwords, each being a program status word (PSW) for a given APU detected reason in this order:

  - arithmetic fault
  - illegal instruction
  - memory controller fault
  - instruction format fault
  - SVC
  - machine malfunction fault

- A single trap block is allocated during OS/32 system generation (sysgen) and is designated in the map under a symbol TBLK1. Any additional blocks can be allocated using the MODULE command at sysgen.

- The trap block has to be patched with zeros for various reasons. If an APU wait is desired, the first word of the pair is set to X'8000' (bit 16 set). The second word does not matter. If custom processing is desired, the first word of the pair is set to the required status and the second word is set to the location where the custom processing begins.

- The APU queue parameter block (QPB) fullword at location QPB.TPTR, (currently X'8' in QPB) has to be patched to the address of the APU trap block after task loading. This patching can be performed using the operator MODIFY command or via a dedicated e-task assembled with the appropriate data structures.

When a task executes on an APU assigned to the patched queue, and a fault is detected for which the PSW in the trap block is not zero, the APU transfers control according to this PSW. In the case of the bit 16 of the first word set in PSW, the APU transfers the task to the CPU ready queue and awaits the task's return. OS/32 will restart the APU when the task is scheduled back to it. OS/32 also restarts the APU when the task for which the APU is waiting is cancelled or terminated.

If the customized processing needs to be done on a per task rather than per queue basis, this can be arranged by patching out the OS/32 code in module APSV routine TMCKAPU that loads the QPB.TPTR into every task control block (TCB) scheduled to the queue. Then, instead of patching QPB.TPTR with the selected task's TCB at location TCB.TPTR (currently X'20' in TCB), it can be patched with the address of the trap block.

3.6 PREVENTING MEMORY ACCESS CONFLICTS

When several processors are executing simultaneously, it is possible for tasks running on two or more processors to require 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 contains 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 both tasks in a Model 3200MPS System can be run 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 at a time can access the buffer.

To ensure that only one task at a time can access BUFLIST, 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.

3.6.1 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 time-out 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
```

3.7 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 setup 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 3-2.

TABLE 3-2 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.

3.8 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 capabilities. However, all the programming facilities available for writing system level control programs are not shown. Table 3-3 summarizes additional facilities and lists the manuals in which they are described.

TABLE 3-3 ADDITIONAL INFORMATION SOURCES FOR MODEL 3200MPS SYSTEM PROGRAMMING

MANUAL	PROGRAMMING METHODS DESCRIBED
Model 3200MPS System Instruction Set Reference Manual	Describes the machine instructions specific to the Model 3200MPS System processor. Also gives a detailed discussion of the APU processor states.
OS/32 Run-Time Library (RTL) Subroutines and Intrinsics Reference Manual	Describes the Perkin-Elmer FORTRAN VII RTL routines available for writing a FORTRAN system level control program that performs the functions described in this chapter.
OS/32 Operator Reference Manual	Describes the operator commands that can be used to perform SVC13 mapping and control functions. APU related functions included APC, LPU, OPTION LPU, QUEUE.
OS/32 Supervisor Call (SVC) Reference Manual	Gives details on how to use SVC6, SVC13 and assembly language programming SVCs.
OS/32 System Macro Library Reference Manual	Describes the time and SVC13 macros.
OS/32 Link Reference Manual	Describes the use of OPTION LPU, APCONTROL and APMAPPING at task linkage time.
OS/32 Application Level Programmer Reference Manual	Gives details on writing a task trap handling routine that can be used to handle APU-related events.
OS/32 System Generation (Sysgen/32) Reference Manual	Describes the use of the MODULE command.

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:

- SVC1
- SVC2 code 7
- SVC3
- SVC6
- SVC7

Intercepting tasks can be loaded and executed under the multi-terminal monitor (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 to be intercepted. For example, a set of buffers allocated for SVC6 interception will be larger than a set of buffers for SVC1 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.

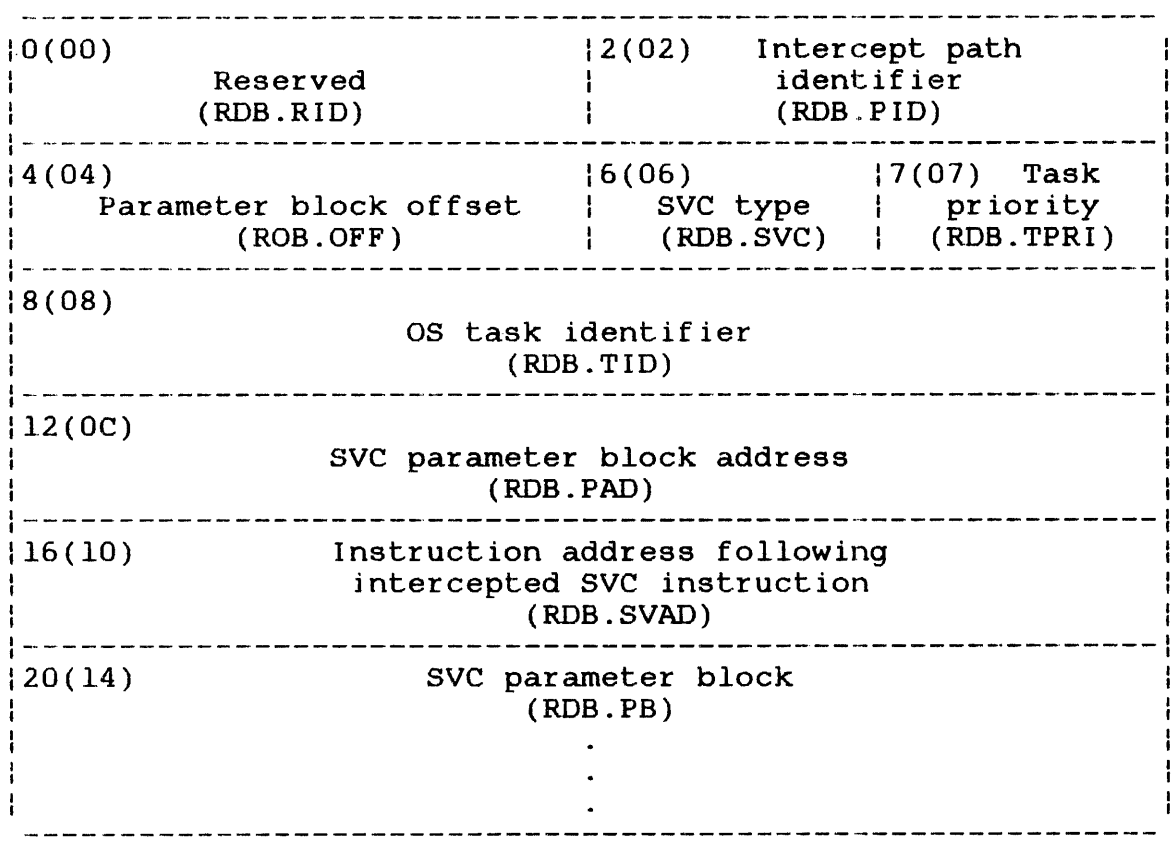


Figure 4-1 Request Descriptor Block

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 SVC1. • 02 indicates SVC2, code 7. • 03 indicates SVC3. • 06 indicates SVC6. • 07 indicates SVC7.
Task priority (RDB.TPRI)	is a 1-byte field containing a decimal number specifying the priority of the task that issued the intercepted SVC.
OS 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 SVC3 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 SVC3 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. See the appropriate Perkin-Elmer Series 3200 Processor User's Manual or the Instruction Set Reference Manual for a more detailed explanation of the standard circular list.

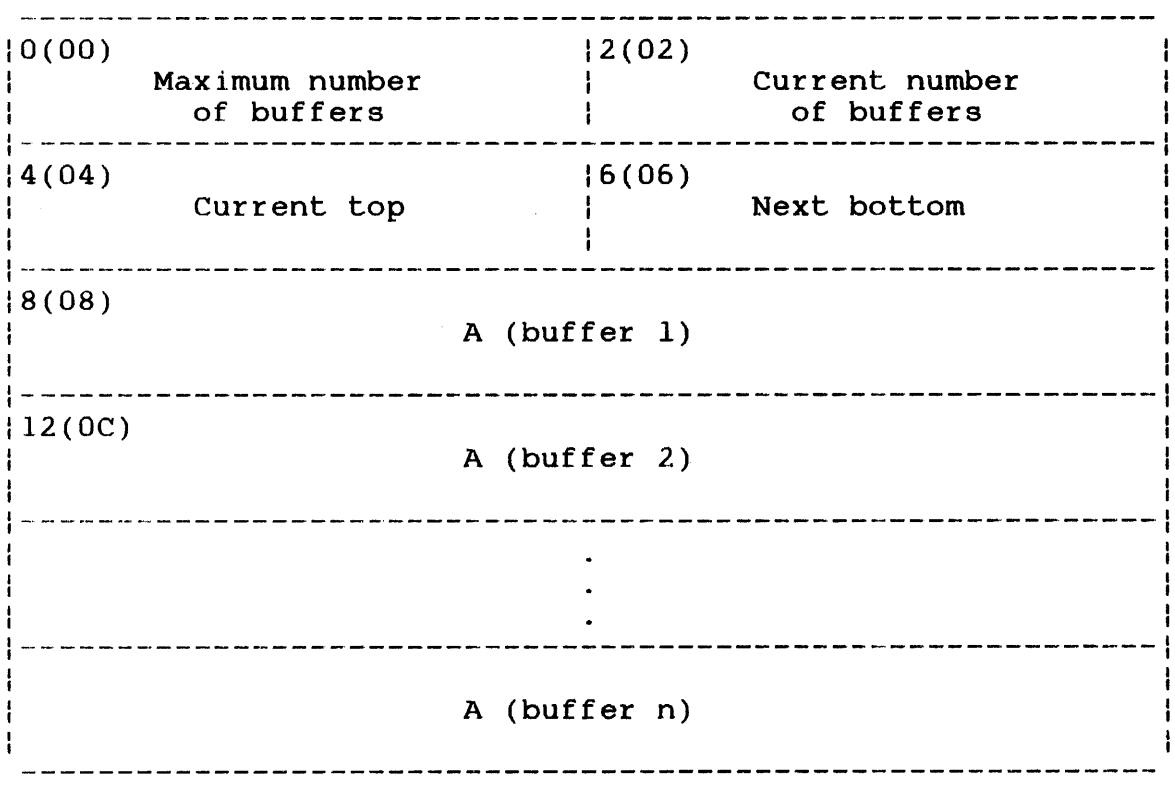


Figure 4-2 System Task Buffer List (Standard Circular List)

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 zero, 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 toward 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 SVC1 or SVC7 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 (fd).

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 SVC1 or SVC7. The recipient nonexistent mode should also be specified. An SVC1 intercept path must be in effect when an input/output 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 SVC6 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 SVC6 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:

SVC1 caller	Any SVC1 issued by the specified task is intercepted.
SVC1 recipient existent	Any SVC1 directed to a logical unit (lu) assigned to the specified device or pseudo device is intercepted. (Note that disk volume interception is not supported for SVC1.)
SVC1 recipient nonexistent	The pseudo device is created, and any SVC1 call specifying an lu assigned to this pseudo device is intercepted.
SVC2 code 7 caller	Any SVC2 code 7 issued by the specified task is intercepted.
SVC2 code 7 recipient existent	This call is invalid.
SVC2 code 7 recipient nonexistent	This call is invalid.
SVC3 caller	If the specified task goes to end of task for any reason, an SVC3 intercept will occur.
SVC3 recipient existent	This call is invalid.
SVC3 recipient nonexistent	This call is invalid.
SVC6 caller	Any SVC6 issued by the specified task is intercepted.
SVC6 recipient existent	Any SVC6 directed to the specified task or pseudo task is intercepted.
SVC6 recipient nonexistent	The pseudo task is created, and any SVC6 call directed to this pseudo task is intercepted.
SVC7 caller	Any SVC7 issued by the specified task is intercepted.

SVC7 recipient existent	Any SVC7 directed to the specified device, disk volume or pseudo device is intercepted.
SVC7 recipient nonexistent	The pseudo device is created, and any SVC7 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:

- Make the task rollable via the IROLL macro. When an SVC is intercepted, the task that issued the SVC is placed in a wait state and made nonrollable. At the discretion of the intercepting task, the application task can be made rollable (assuming the application task can be rolled).

- Allow the application task to execute while it processes a proceed SVC via code built by the IPROCEED macro. When an SVC is intercepted, the application task that issued the SVC is placed in a wait state and made nonrollable. At the discretion of the intercepting task, the application task that issued the intercepted SVC can proceed with its execution while the intercepting task processes the SVC.

- Obtain data from the application task memory space via the IGET macro.

- Write data into the writable memory space of the application task via the IPUT macro.

- Send a task queue trap to the application task via the ITRAP macro. While processing the SVC, the intercepting task may find it necessary to send a task queue trap to the application task. The task queue item sent must have a valid OS/32 reason code in the high-order byte. In addition, the TSW of the application task must have the task queue entry (TQE) bit associated with the reason code set.

A monitor control intercept path allows the intercepting task to be notified whenever one of the designated SVCs is issued by an application task. Monitor control differs from full control in that once OS/32 has sent the task event trap to the intercepting task, the SVC is passed to the appropriate OS/32 executor and the task that issued the SVC proceeds with normal processing.

The following guidelines should be followed when assigning a level of control to the intercept path:

- Only monitor control can be specified for SVC3 intercept paths. Either full or monitor control can be specified for all other SVC type intercept paths.
- Only one full control intercept path can be attached to a device or task (or pseudo device or task) for each type of SVC to be intercepted.
- A task or device (or pseudo task or device) can be attached to any number of monitor control intercept paths.

Example:

```
            ICREATE NAME=DEVNAME,MODE=RX,CONTROL=FC,SVC=(7)
            ICREATE NAME=DEVNAME,MODE=RX,CONTROL=FC,SVC=(1)
            ICREATE NAME=DEVNAME,MODE=RX,CONTROL=MC,SVC=(7)
            ICREATE NAME=DEVNAME,MODE=RX,CONTROL=MC,SVC=(1)

DEVNAME     DC C'        '  DEFINE 8 BLANK CHARACTERS

            DC C'MAG '

            DC C'        '  DEFINE 8 BLANK CHARACTERS (FD)

            DC C'        '  DEFINE 4 BLANK CHARACTERS (EXTENSION)
```

In this example, a full control SVC7 intercept path is attached to device MAG:. A full control SVC1 intercept path is also attached to MAG:. No other SVC1 or SVC7 full control intercept paths can be attached. Of course, any number of SVC1 and SVC7 monitor control intercept paths can be attached to MAG:; here, one SVC7 and one SVC1 monitor control paths are attached.

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

SVC1 and SVC7 can be intercepted during end of task processing (including end of task processing after cancel), if intercept paths exist from these SVCs to devices assigned to the task's logical units. The intercepting task must be careful when writing into the operating system address space while executing these SVCs so as not to destroy the system's integrity.

If the application task is cancelled while the intercepting task is processing the SVC, SVC processing is aborted and the application task proceeds to end of task.

4.10 TERMINATING THE INTERCEPTED SUPERVISOR CALLS (SVCs)

When the intercepting task receives an SVC from a full control intercept path, the intercepting task has the option of returning the SVC to the operating system for processing. To do this, the intercepting task executes code built by an ICONT macro that allows the operating system to resume processing the intercepted SVC as if the intercept had never occurred. The ICONT macro cannot be used if an IPROCEED or IROLL macro has been used.

If the intercepting task chooses to process the SVC, the intercepting task executes code built by an ITERM macro after the SVC is processed. ITERM terminates the interception and, if no IPROCEED has been issued, allows the application task to resume execution with the instruction immediately following the intercepted SVC instruction.

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 not.	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

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

ERROR CODE	MEANING	RELEVANT MACROS
ST	Invalid state for call; e.g., IROLL followed by ICONT or issuing INPUT 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
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
symbol	ICREATE	SVC={(1) / (2,7) / (3) / (6) / (7)}
		,MODE={CL / RX / RN}
		,NAME=pointer
		,TID=pointer
		,CONTROL={FC / MC}
		,BUFFERL=pointer
		[,HANDLER=pointer]
		,PID=pointer
		,EXEC=pointer
		[,PBSIZE=n]
		[,SVAR=pointer]
		[,ERROR=pointer]
		[,PCB=pointer]
		[,FORM=L]
		[,NINTC=n]

Operands:

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

- (1) indicates SVC1
- (2,7) indicates SVC2 code 7
- (3) indicates SVC3
- (6) indicates SVC6
- (7) indicates SVC7

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

Operands:

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

- (1) indicates SVC1
- (2,7) indicates SVC2 code 7
- (3) indicates SVC3
- (6) indicates SVC6
- (7) indicates SVC7

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 SVC1 or SVC7, or a pseudo task is created for SVC6. 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 SVC7 interception, the requested access privileges are ignored and shared read/shared write privileges are granted. If an SVC1 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 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			
SVC=	MODE=	NAME=	FUNCTION
(1)	CL	taskid	Intercepts any SVC1 issued from the task.
	RX	fd	Intercepts any SVC1 directed to the existing device.
	RN	fd	Creates a pseudo device and intercepts any SVC1 directed to it.
(2,7)	CL	taskid	Intercepts any SVC2 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.
(6)	CL	taskid	Intercepts any SVC6 issued from the task.
	RX	taskid	Intercepts any SVC6 directed to the existing task.
	RN	taskid	Creates a pseudo task and intercepts any SVC6 directed to it.
(7)	CL	taskid	Intercepts any SVC7 issued from the task.
	RX	fd	Intercepts any SVC7 directed to the existing device.
	RN	fd	Creates a pseudo device and intercepts any SVC7 directed to it.

TID= indicates the address of a fullword location containing a taskid. This parameter, which is mutually exclusive with the NAME= parameter, can be used when MODE=CL, or MODE=RX with SVC6, 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 a 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 SVC2 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]

Operands:

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; requests made after TERM=CS is issued cannot 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 that 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]

Operands:

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]

Operands:

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]

Operands:

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]

Operands:

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 program status word (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]

Operands:

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={Y / N}
		[,ERROR=pointer]
		[,PCB=pointer]
		[,FORM=L]
		[,CC=n]

Operands:

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 that 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]

Operands:

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 that 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]

Operands:

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, one 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 that 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 SVC1 to the existing real device MAG1. Each time an SVC1 is issued to MAG1, the program prints the following message:

```
SVC 1 CALL INTERCEPTED
```

The SVC1 is terminated with a device unavailable error code (X'A0').

```
        $RDB                    DEFINES AN RDB STRUCTURE

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

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

        ABL     0,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 user task (u-task) can assign and write. The user's data buffer is passed to the OS/32 command processor via SVC2 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 u-task            *
*        can assign and write.  The user's data buffer is               *
*        passed to the operating system command processor via           *
*        an SVC2 code 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 to 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 bidirectional input/output control (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 specification 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., a 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 (LF) pending state for the next write operation.

For drivers that support both input and output operations (e.g., a 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 an LF is output. Two types of LF 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 and then performs a VFC operation if it is in LF pending state before the character is echoed.

If a driver uses the hardware-echo feature and is in LF 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.

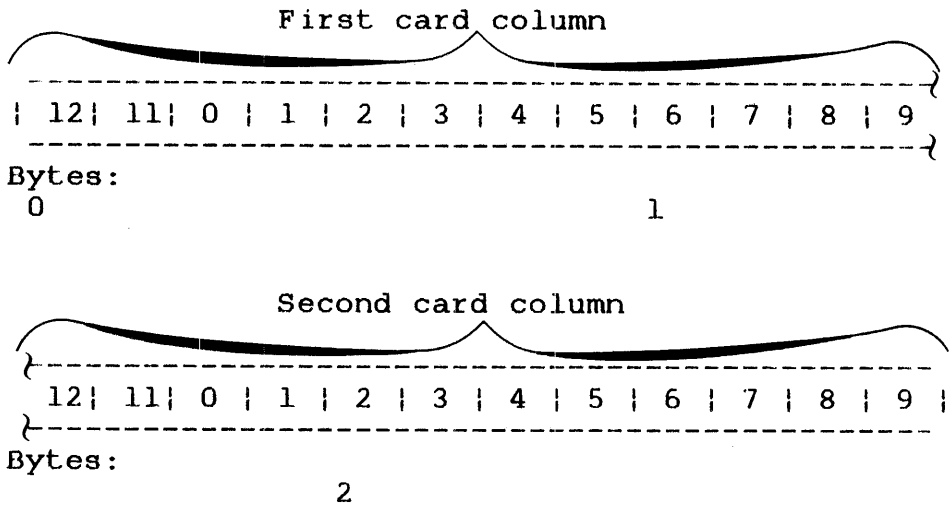
5.3 CARD READERS

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.

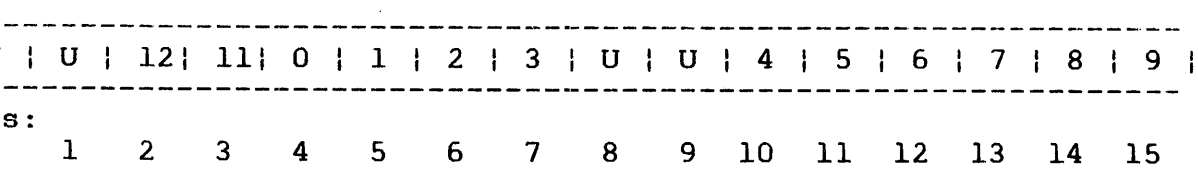
During read binary operations, each pair of card columns (12 bits each) is unpacked into three bytes having the following format.

Read Binary Format:



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

Read Image Format:



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.

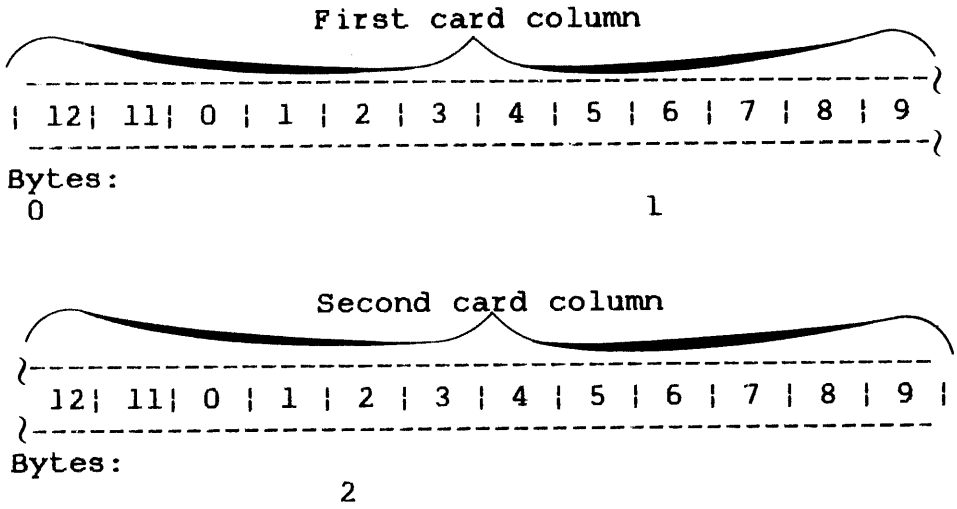
5.4 CARD READER/PUNCH DEVICES

Card reader/punch devices supported by Perkin-Elmer 32-bit processors 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.

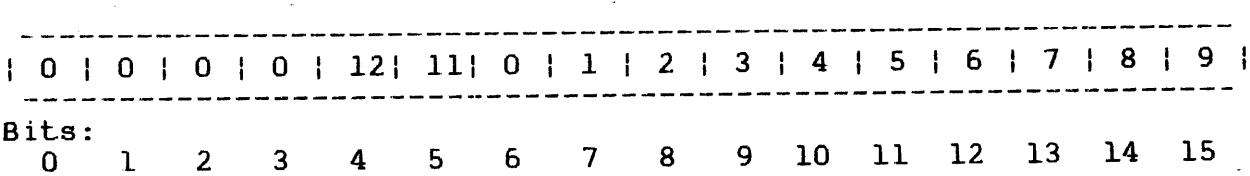
During read binary operations, each pair of card columns (12 bits each) is unpacked into three bytes having the following format.

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:

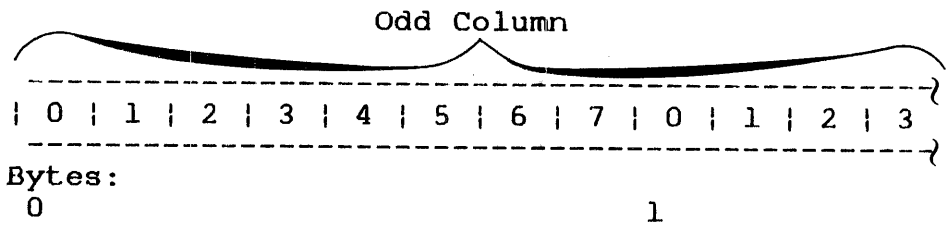


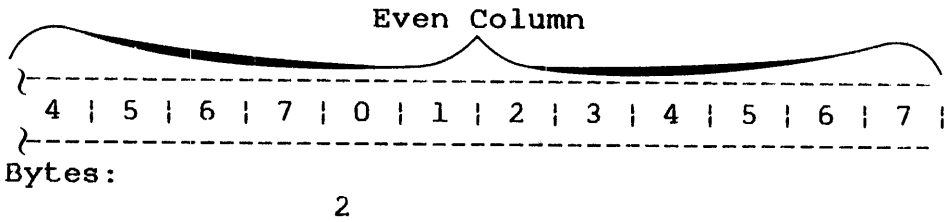
During write ASCII operations, each byte of data is translated from ASCII into a 12-bit Hollerith code. Depending on the device code chosen, the following can occur:

- 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.

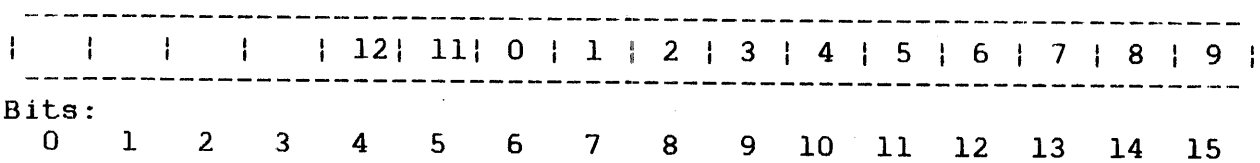
Write Binary Format:





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:



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 (CR) 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 a leader. The user data is output until the buffer is empty, or a CR 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 CR 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 CR is found in the data stream. An LF is automatically appended to the detected CR; or no CR 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 CRs or LFs. On image read, a 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 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. CR 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' through X'84' and X'95' through 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) CR 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 CR 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 (10ms for read; 30ms 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 SVC1 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 SVC1 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 SVC1 and SVC7, 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 can 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 SVC1. To perform gapless I/O to a magnetic tape, a task must issue an SVC1 call that specifies, among other things, a pair of buffer queues, the IN-QUEUE and the OUT-QUEUE. The driver takes buffers from the 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 SVC1 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 executive tasks (e-tasks), privileged user tasks (u-tasks), and diagnostic tasks (d-tasks), linked with bare disk privileges (OPTION DISC), can access a bare disk. Nonprivileged 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, 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 CR 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 CR is found in the data stream. An LF is automatically appended to the detected CR; if no CR 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 CRs or LFs. 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 (MUX)

ASCII operations are not supported by the Perkin-Elmer digital MUX. 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 SVC1 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 (AIC)

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

The random address field of the SVC1 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.

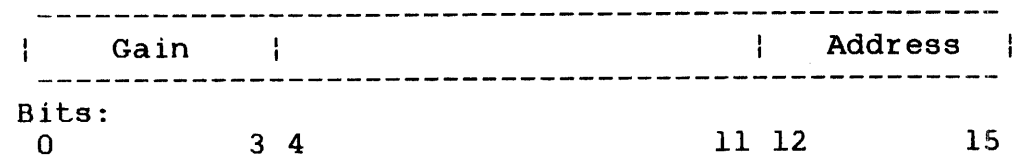


Figure 5-1 Random Field Format

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 AIC is a halfword device. This complies with the Instrument Society of America (ISA) definition of sequential analog input.

5.18 ANALOG OUTPUT CONTROLLER (AOC)

All command functions are ignored by the Perkin-Elmer AOC. 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.

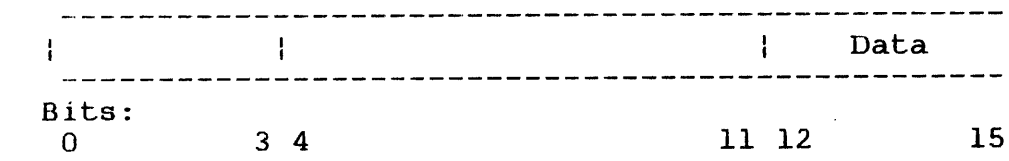


Figure 5-2 Analog Output Data Format

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 AOC 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 time-out 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 SVC1 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 SVC1 parameter block points to a buffer (K1) consisting of image halfwords for transfer to an output device. The random address field of the SVC1 parameter block points to another buffer (K2) of halfwords designating masks, each of which defines the corresponding bit position of the halfwords in K1 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:

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

Where:

$\bullet$	means logical AND.
+	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 SVC1 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 power input/output (PIO) controller is a halfword device.

5.20 ETHERNET DATA LINK CONTROLLER (EDLC)

Ethernet is a multi-access, packet-switched communication system for carrying data among locally distributed computing systems. The EDLC provides processor-to-processor serial communication clocked at 10Mb per second over a common coaxial cable up to 500 meters long. The Ethernet specifications will support over 100 processor interfaces in a variety of configurations.

The following publications provide more information:

- OS/32 Network Drivers Programming Reference Manual
- Perkin-Elmer Series 3200 Ethernet Controller Installation, Operation and Programming Manual
- Perkin-Elmer Series 3200 Ethernet Interface Test Program Description
- Interlan NM10A Ethernet Protocol Module Users Manual
- Ethernet Network Specification Document

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

TYPE	DEVICE	ATTRIBUTES								
		RD	WRT	BTS	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 (TTY) 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
TTY 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
	Perkin-Elmer Carousel 300 ^ $	x	x			x			x	x

TYPE	DEVICE	ATTRIBUTES								
		RD	WRT	BTS	BIN	WAT	RND	FLP	INT	HLT
TTY keyboard printer (Continued)	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 bits per inch (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		
	9-track, 75 inches per second (ips) 800 bpi	x	x		x	x		x		

TYPE	DEVICE	ATTRIBUTES								
		RD	WRT	BITS	WAIT	RND	FLP	INT	HLT	
Magnetic tape (Continued)	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 head per track (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			
	50Mb CDD50, fixed portion (25Mb)	x	x	x	x	x	x	x		
	50Mb CDD50, removable portion (25Mb)	x	x	x	x	x	x	x		
	67Mb disk, fixed	x	x	x	x	x	x			
	67Mb disk, removable	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			

TYPE	DEVICE	ATTRIBUTES								
		RD	WRT	BTS	WIN	RAT	FND	ILP	HNT	LLT
Disks (Continued)	68.6Mb disk mass storage media (MSM) 300 disk system (256Mb formatted)	x	x	x	x	x	x			
		x	x	x	x	x	x			
	MSM 80 disk system (67Mb formatted)	x	x	x	x	x	x			
	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 (VDUs)	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
	Model 6100 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 (MUX)	Digital MUX controller	x	x		x	x	x			

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

* CLI - Current loop interface
^ CLCM - Current loop communications multiplexor
$ RS-232C

ATTRIBUTES

RD	Read
WRT	Write
TS	Test and 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
0B	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

T

PERKIN-ELMER

PUBLICATION COMMENT FORM

We try to make our publications easy to understand and free of errors. Our users are an integral source of information for improving future revisions. Please use this postage paid form to send us comments, corrections, suggestions, *etc.*

1. Publication number _____

2. Title of publication _____

3. Describe, providing page numbers, any technical errors you found. Attach additional sheet if neccessary.

   _____

   _____

   _____

4. Was the publication easy to understand? If no, why not?

   _____

5. Were illustrations adequate? _____

   _____

   _____

6. What additions or deletions would you suggest? _____

   _____

   _____

7. Other comments: _____

   _____

   _____

From _____ Date _____

Position/Title _____

Company _____

Address _____

_____

_____

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

Data Systems Group
106 Apple Street
Tinton Falls, NJ 07724

ATTN:
TECHNICAL SYSTEMS PUBLICATIONS DEPT.

FOLD

FOLD

STAPLE

STAPLE