November 1978

RSX–11M
System Logic Manual

Order No. AA-5579A-TC

VOLUME 1

RSX-11M V3.1

To order additional copies of this document, contact the Accessories and Supplies Group, Product Line 86, Digital Equipment Corporation, Cotton Road, Nashua, New Hampshire 03060.

digital equipment corporation · maynard. massachusetts

First Printing,November 1978

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL
DEC
PDP
DECUS
UNIBUS
COMPUTER LABS
COMTEX
DDT
DECCOMM
ASSIST-11
VAX
DECnet
DECsystem-10
DECtape
DIBOL
EDUSYSTEM
FLIP CHIP
FOCAL
INDAC
LAB-8
DECSYSTEM-20
RTS-8
VMS
IAS
MASSBUS
OMNIBUS
OS/8
PHA
RSTS
RSX
TYPESET-8
TYPESET-11
TMS-11
ITPS-10
SBI

CONTENTS

VOLUME I

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS

VOLUME II

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

Page

CONTENTS (Cont.)

FIGURES

CONTENTS (Cont.)

CONTENTS (Cont.)

PREFACE

MANUAL OBJECTIVES AND READER ASSUMPTIONS

This manual is intended for the experienced system programmer; one who is familiar with RSX-11M operation and has an acquaintance with the RSX-11M Executive code. The manual presents information tutorially. However, the manual also contains a great deal of cross-reference information for very experienced programmers who may not need the tutorial information.

The System Logic Manual primarily discusses the Executive. However, cross-references for MCR and File Control Processor are also included.

PREREQUISITE MANUALS

The reader and user of this manual is expected to have read and understood the contents of the following manuals:

Introduction to RSX-11M

IAS/RSX-11 MACRO-11 Reference Manual

RSX-11M Executive Reference Manual

RSX-11M Task Builder Reference Manual

RSX-11M Guide to Writing an I/O Driver

RSX-11M Operator's Procedures Manual

RSX-11M System Generation Manual

THE STRUCTURE OF THE LOGIC MANUAL

VOLUME 1

Chapter 1 - Introduction
Chapter 1 presents a general description of the RSX-11M V3.1 system for those who may need a fundamental introduction. This chapter expands much of the material found in Introduction to RSX-11M.

Chapter 2 - Memory Resource Allocation

Chapter 2 describes Memory Management and the important routines that allocate memory resources. Specifically, Chapter 2 describes the Loader, the Shuffler, $NXTSK, $CHKPT, and associated routines.

Chapter 3 - Interrupt Processing

Chapter 3 describes Executive interrupt and fork processing. The important interrupt routines are described. A figure along with supporting text describes the code path followed by the interrupt and fork routines that a sample driver uses.

Chapter 4 - Privileged Tasks

Chapter 4 describes the purpose of privileged tasks, their use and the cautions concerning their use, the use of $SWSTK (switch stack), and mapping for both privileged and nonprivileged tasks.

Chapter 5 - MCR Interface

Chapter 5 describes the function of the MCR interface, the MCR Dispatcher, and how MCR processes a command line entered at a terminal.

Chapter 6 - I/O Processing

Chapter 6 discusses the internal processing of the QIO directive.

Chapter 7 - Module Descriptions

Chapter 7 contains brief descriptions of the modules that make up the Executive. Entry points, inputs, outputs, and exit status of the modules are also described.

Chapter 8 - Data Areas and Control Blocks

Chapter 8 contains figures that show various system linkages in a generalized form. Important functions of the Device Control Block, Unit Control Block, and Status Control Block are also discussed. Also shown are all the bit definitions for the system control blocks.

Chapter 9 - Cross-references

Chapter 9 contains important cross-references, created by the CREF program, that you can use to find your way through the system listings. The references include:

For the Executive:

- Module-to-routine cross-references
- Symbol-to-module cross-references

For MCRMU (multiuser MCR):

- Symbol-to-module cross-references (created by the CREF program)
- MCRMU segment cross-references
- SYS symbol-to-module cross-references
- SYS segment cross-references

For the File Control Processor (BIGFCP):

- Symbol-to-module cross-references
- BIGFCP segment cross-reference

Chapter 9 also contains cross-references between Executive modules and conditional assembly parameters that these modules contain.

VOLUME 2

Appendix A - RSX-11M Supported Devices
Appendix A contains a list of devices supported by RSX-11M V3.1.

Appendix B - Coding Standards and Conventions
Appendix B contains an explanation of the coding standards and conventions that RSX-11M follows. Appendix B also describes the Executive's use of co-routines by explaining an example co-routine from Executive code.

Appendix C - Macro Expansions
Appendix C contains the expansions of all the macros used in the system code.

Appendix D - Listing of Conditional Assembly Parameters
Appendix D lists all the conditional assembly parameters and their meanings.

Appendix E - General Fault Isolation
Appendix E contains a generalized approach to program fault isolation for RSX-11M.

Appendix F - System Tuning
Appendix F contains many ideas that can help you to improve system performance.

CHAPTER 1

INTRODUCTION TO THE RSX-11M V3.1 OPERATING SYSTEM EXECUTIVE

This introduction is a tutorial for those who are beginning to learn the RSX-11M Executive internal logic. However, this manual assumes that you have at least read and understood the RSX-11M Introduction, the RSX-11M Operator's Procedures Manual, the RSX-11M Task Builder Reference Manual, and the RSX-11M System Generation Manual. If you are familiar with the RSX-11M Executive or you are an experienced system programmer, you may want to begin this manual with Chapter 2, which assumes that you have a basic knowledge of the Executive and describes the memory structures of RSX-11M.

1.1 RSX-11M SYSTEM

RSX-11M is a real-time operating system. This means that RSX-11M responds quickly to input conditions or input data. RSX-11M is also a multiprogramming system. This combination allows real-time activity (for example, process control) to occur along with program development (interactive terminals) and other user jobs. At one extreme, RSX-11M can be a dedicated process control system, and at the other, a system for developing and running applications programs.

1.2 SYSTEM GENERATION

RSX-11M offers a wide range of services and utilities from which to choose. Each installation selects from these options to shape its version of RSX-11M according to the processor and peripherals available and the purpose of the system. You perform a system generation (SYSGEN) process to select these options.

Every installation intitially receives an RSX-11M system on distribution media. You run this system and use its resources to generate a target system configured to your installation's needs.

System generation is done in two phases. During the first phase SYSGEN defines and assembles the Executive (the kernel or "brain" of the operating system that responds to external requests) by conducting a dialogue with you. Query programs pose questions at a terminal. Your answers to the questions determine the Executive service options, processor options, and peripheral devices to be incorporated into the system. During the second phase, SYSGEN builds the Executive, allows you to define memory structures called partitions, and builds and installs the system programs.

You complete the SYSGEN process by saving and bootstrapping the new system. Saving a system means writing the image of an RSX-11M system that has been resident in main memory into the system image file from which it was bootstrapped. You do this with the Save command, which saves the image to allow a hardware bootstrap or the Boot command to later reload and restart the system.

You bootstrap (boot) a system by either using the switches on the processor control panel or using the Boot command. The Boot command bootstraps a system that exists as a system image file on a Files-11 (the RSX-11M file structure) volume. The Boot command immediately terminates the system in operation and starts another. The Save command, the Boot command, and the process of booting a system with the switches are all described in the RSX-11M Operator's Procedures Manual.

To change either the hardware or software configuration of an installation, you must perform another system generation. The RSX-11M System Generation Manual describes the system generation process in detail.

1.3 MAJOR COMPONENTS OF RSX-11M

RSX-11M requires the organized interaction of the following components:

- Memory resource management. Memory is the processor storage medium in which loaded user programs, the Executive , and control blocks of data reside. Much of the Executive's work involves memory resource management and control.

- Task scheduling and processing. Tasks are system or user programs that perform needed functions and manipulate data to achieve some goal. The Executive controls task processing and handles specific requests issued by the tasks.

- Interrupt processing. The Executive processes synchronous and asynchronous events that occur as a result of task processing. Examples of these events include software errors, I/O completion, illegal instructions, and power failure.

1.4 MEMORY

1.4.1 Memory Partitions

A partition is a continuous area of memory in which executable programs called tasks can be run. The typical memory organization consists of an area for the Executive and areas for system- or user-controlled partitions. A partition has the following characteristics:

- A name

- A defined size

- A fixed base address

- A defined type

1.4.2 Partitions In Mapped And Unmapped Systems

RSX-11M runs on almost all models of the PDP-11 processor. The PDP-11 addressing scheme allows a program to address directly only 32K words of memory. For larger memories, DIGITAL has a KT-11 Memory Management Unit (hardware) available for all models of the PDP-11 except the PDP-11/03/04/05/10/20 processors. The KT11 Memory Management Unit associates addresses expressed in programs ("virtual" addresses in the range 0 to 32K) with actual locations in memory ("physical" addresses). Physical addresses can range from 0 to 124K words on all processors other than the PDP-11/70. Physical addresses on a PDP-11/70 can range from 0 to 1920K words.

Mapping is the process that associates virtual addresses with physical addresses. Therefore, a PDP-11 system that includes a KT11 Memory Management Unit is called a mapped system. Conversely, systems without a KT11 are called unmapped systems. In a mapped system, a task can be installed in any system partition or user partition large enough to contain it. In an unmapped system, the task is bound to physical memory and must be installed in the partition that starts at the same memory address as the partition for which it was built.

Whether a system is mapped or unmapped affects the way in which you create tasks. Before a compiled program (object code) can be run, it must be processed by the Task Builder program (linker). The Task Builder produces a task image that runs in a memory partition.

If a system is unmapped, you must specify to the Task Builder the base address of the partition in which the task is to be run. You cannot run the resulting task in a partition that has a base address different from the address you specified to the Task Builder.

In a mapped system, however, every task (other than a privileged task mapped into the Executive) has a virtual base address of 0. Transparently to the user, the KT11 maps the virtual addresses of a task to the actual physical addresses in which the task resides. A task in a mapped system can therefore run in any partition large enough to contain it.

You need not rebuild nonprivileged tasks in a mapped system when physical partition boundaries move. This is true because nonprivileged tasks on a mapped system run at a virtual base address of 0, rather than at a physical base address.

If you move the symbols that are referenced in the code, you must rebuild privileged tasks in either system because they are linked to the Executive symbol table file. You may be required to rebuild nonprivileged tasks only if you change any of the task's attributes such as checkpointability. The task's attributes can be changed when you use the Install command to install the task. You use the Task Builder to establish the attributes when building a task. Consult the RSX-11M Task Builder Reference Manual for a comprehensive discussion of task attributes and associated Task Builder switches. See the RSX-11M Operator's Procedures Manual for a description of the Install command.

1.4.3 Partition Types

RSX-11M supports two types of partitions in which tasks can execute:

1. System-controlled

2. User-controlled

In a system-controlled partition, the Executive allocates available space to accomodate as many tasks as possible at any one time. This allocation may involve shuffling resident tasks to arrange available space into a continuous block large enough to contain a requested task. The Shuffler, which is a privileged task and a SYSGEN option, shuffles the tasks and memory space to make the needed space for the requested task. Only mapped systems support system-controlled partitions.

A user-controlled partition is exclusively allocated to one task at a time. Both mapped and unmapped systems support this type of partition.

1.4.4 Subpartitions

You can subdivide a user-controlled partition into as many as seven nonoverlapping subpartitions. Like its parent main partition, a subpartition can contain only one task at a time. Because the subpartitions occupy the same physical memory as the main partition, tasks cannot be simultaneously resident in both the main partition and one of its subpartitions. However, because each subpartition can contain a task, up to seven tasks can potentially run in parallel within a main partition.

The purpose of subpartitioning is to reclaim large memory areas in unmapped systems. For example, when a large task that requires a main partition is either no longer active or can be checkpointed (written out to a disk to make room for a higher priority task), subpartitioning allows a number of smaller tasks to use the partition space.

1.4.5 Memory Structure

RSX-11M memory in a typical system can be divided into the following parts:

- The Executive, which consists of:

  - Trap vectors. The trap vector area contains the hardware and interrupt vectors; it requires 128 words. During SYSGEN, you can expand this area to 256 words.

  - System stack. The system stack area is an internal storage area for Executive use. The Executive uses it for nesting interrupts, saving registers and data, and internal calls. The stack requires 60 to 110 words depending upon options selected at system generation time.

  - System common data. This area contains system pointers that are filled in during system generation and used by the Executive and privileged tasks during execution.

  - The Executive code. The Executive coordinates and manages system resources and processes specialized system functions. System generation options determine the size and abilities of the Executive.

- Dynamic Storage Region (DSR). The Executive continually uses temporary storage in memory. The Executive acquires, uses, and then returns the memory that it used to the available memory pool. If a given Executive service routine requests dynamic storage and it is unavailable, the Executive informs the user task, which usually waits for some memory to become available. The size of this region is important. If it is too small, long waiting periods or system deadlocks can occur. If it is too large, fewer tasks can fit into the remaining memory. The size of the region is a system generation parameter.

  You can extend the initial allocation of dynamic storage on line by issuing the MCR command, Set /Pool, from the console. However, the use of this command is limited in that this expansion can occur only into space that is not being used. This space, if it exists, is between the Dynamic Storage Region space and the first partition of memory.

● Device drivers:

You can include three drivers in the 8K Executive during SYSGEN:

1. A disk driver

2. A cassette, DECtape, magtape, line printer, or floppy disk driver

3. A terminal driver

In general, Executives larger than 8K contain additional resident drivers which you include during system generation. Some drivers can be made loadable; that is, they reside on disk and are loaded into memory when they are needed. Therefore, loadable drivers save memory space because they occupy memory only when needed and they do not occupy Executive virtual address space.

● Loader:

The Loader is a task that runs in its own partition, which is resident in the Executive. Thus, it can run in parallel with system and user tasks. The Loader, which is device independent:

1. Loads tasks upon initial load requests

2. Writes checkpointable tasks to disk when required (see checkpointing, in this chapter)

3. Reloads previously checkpointed tasks when memory becomes available, allowing them to actively compete for processor resources.

● MCR and TKTN tasks:

- The Monitor Console Routine (MCR) processes system commands that you enter at a terminal. These commands are directed to the MCR processor. MCR either executes the commands itself, or activates a system or user-written task that can service the commands.

- The Task Termination Notification Task (TKTN) performs two functions:

  1. It prints out messages and tries to print the contents of the registers of a task that has been aborted due to an error.

  2. It prints out messages for device drivers.

  Ideally, TKTN runs either in a partition in which all tasks are checkpointable or execute quickly, or in its own partition. The reason for this is that TKTN must be in memory in order to print messages. If TKTN cannot get memory space to execute, the Executive queues up messages to TKTN, thereby using up Dynamic Storage Region space. It is conceivable that all the Dynamic Storage Region could be used up for this purpose; this would cause the system to hang up.

- The file system:

  Files-11 is a system of formatting files that are held on volumes. Files-11 volumes are magnetic media (tapes or disks) that have been specially formatted by the MCR command, Initialize Volume. Volumes that are not properly formatted are considered to be "foreign." RSX-11M includes a file exchange utility that translates files in DIGITAL's DOS or RT-11 format into Files-11 format.

  Your tasks that run on RSX-11M access data within files on Files-11 volumes through the use of two sets of subroutines:

  - File Control Services (FCS)

  - Record Management Services (RMS)

  Both FCS and RMS provide the ability for your tasks to perform record- or block-I/O operations on Files-11 volumes. FCS and RMS are system interfaces between the I/O programs that you write and the files on the Files-11 volumes that you want to access. These interfaces provide device independence and allow you to take advantage of different methods of file organization.

  FCS imposes a single logical organization on your files. This logical organization is called the sequential file organization and FCS imposes it on all files regardless of medium.

  In contrast to FCS, RMS provides three file organizations - sequential, relative, and indexed.

  The MACRO-11 I/O programming that you do differs between FCS or RMS. Therefore, you must become familiar with the contents of the manuals that describe each one. The respective manuals are:

  For FCS:

  - <u>IAS/RSX-11 I/O Operations Reference Manual</u>

  - <u>RSX-11M I/O Drivers Reference Manual</u>

For RMS:

- Introduction to RMS-11
- IAS/RSX-11M RMS-11 MACRO Programmer's Reference Manual

The Files-11 Ancillary Control Processor (F11ACP) is a group of Executive subroutines that process and control the I/O control structures and devices for RMS or FCS. The Executive, FCS, and RMS use F11ACP; however, its operation is transparent to you.

F11ACP is available in three versions. The first and smallest (FCPNMH.TSK) requires 2K of memory. FCPNMH does not support multi-header files or RMS record blocking. The second (FCP.TSK) requires 2.5K of memory. The third version (BIGFCP.TSK) requires from 4.5K to 8K of memory. You select these versions during SYSGEN. The RSX-11M System Generation Manual fully describes these versions, the reasons for their use, and the methods of installation.

- The print spooler:

  The print spooler task (PRT) speeds up the operation of MACRO-11, the Task Builder, and compilers because they do not have to wait for I/O to complete on the relatively slow line printer. Instead, the listing files are written to a disk. Subsequently, PRT prints the files as they appear in a queue.

  Any task that uses the line printer to print files may use the print spooler. For example, the RSX-11M Peripheral Interchange Program (PIP) can optionally use the PRT task to print files.

- User task partitions

  User tasks run in the remaining memory in the memory structure. The partitions and tasks can be configured to the system user's requirements.

1.4.6 Example Of A 16K Unmapped System

Figure 1-1 illustrates the memory layout of a sample 16K unmapped system. The Executive region, which requires 8K, consists of the Executive and the user-controlled main partition named SYSPAR. This partition contains the file system (FCPNMH), the Monitor Console Routine (MCR), and the Task Termination Notification routine (TKTN). The file system is checkpointable and has a lower priority than MCR or TKTN. Therefore, if the file system is running and a system user requests MCR, the Executive checkpoints the file system and loads and starts MCR.

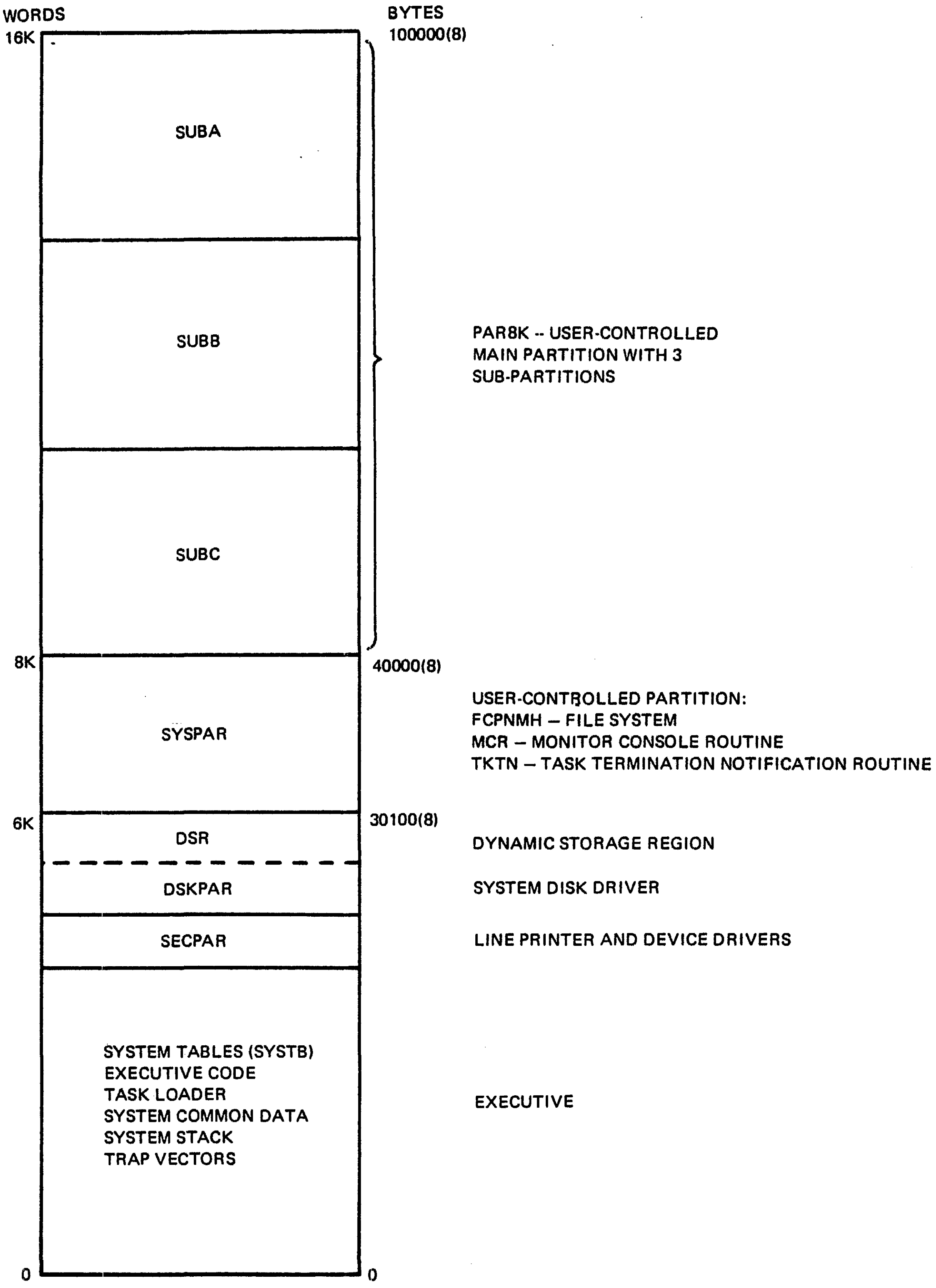


Figure 1-1 Sample Unmapped 16K System Memory Layout

The user area contains a user-controlled main partition named PAR8K, 8K in length. PAR8K contains three subpartitions, named SUBA, SUBB, and SUBC. Language processors and the Task Builder use the 8K partition for program preparation. These programs usually have a low priority and may be checkpointable.

The three subpartitions are available for real-time tasks. A task in the main partition is checkpointed if:

- It is checkpointable
- Another higher priority task needs the partition, or a subparition

If tasks occupy the partitions SUBA, SUBB, SUBC, and SYSPAR and the tasks are ready to run, the Executive gives CPU resources to the task with highest priority.

1.4.7 Example Of A Mapped 124K-word RSX-11M System

Figure 1-2 is an example of a large mapped system.

Besides the Executive, the system contains DRVPAR, which is a system-controlled partition for loadable device drivers including the terminal driver. Loadable drivers residing on a disk are loaded by a user command when they are needed.

SYSPAR is a 2K user-controlled partition that contains the Monitor Console Routine Multi-user (MCRMU) task, TKTN, and the Shuffler task. The Shuffler is discussed later in this chapter.

FCPPAR is a 6K partition for the primary file control system, BIGFCP. The 6K size is sufficient to allocate approximately 50 file control blocks (FCBs).

All other tasks run in the system-controlled GEN partition.

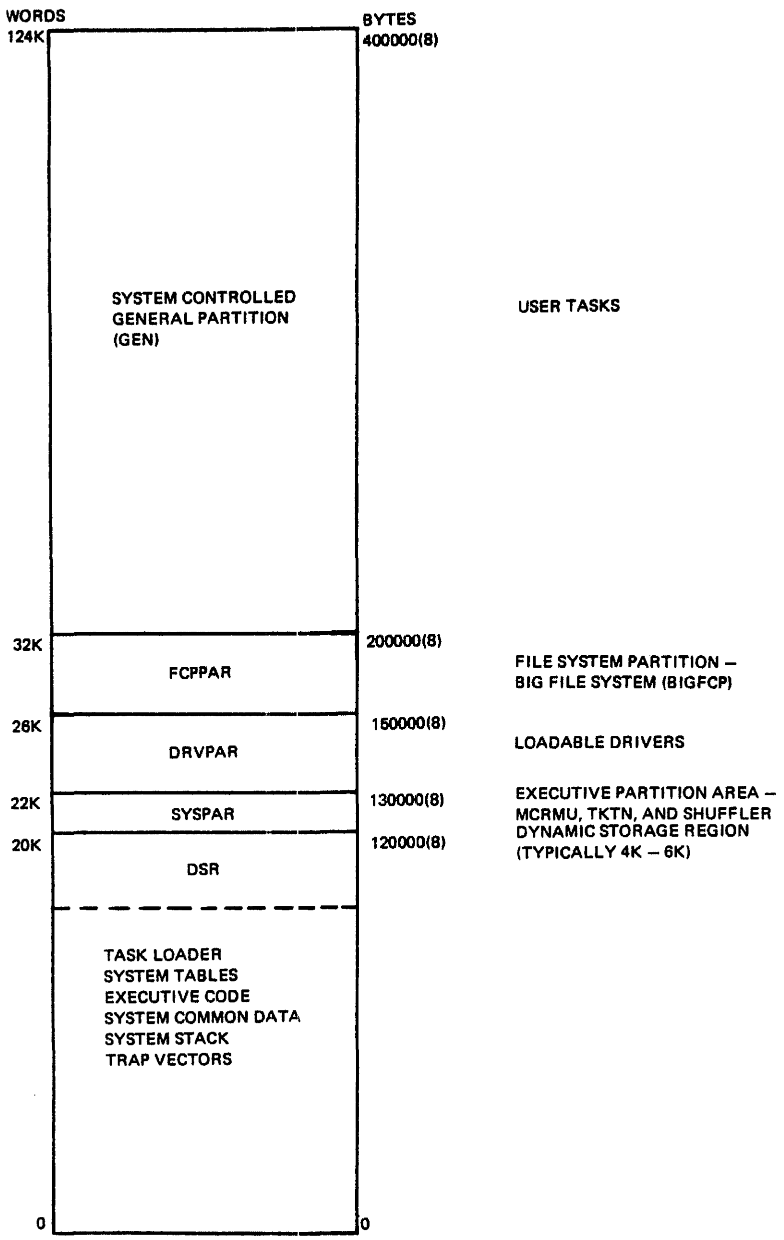


Figure 1-2 Example of a Mapped 124K RSX-11M System

1.5 TASK PROCESSING

To make a task known to the Executive, you install it. When you install a task (by issuing the Install command; see the RSX-11M Operator's Procedures Manual), the system records a number of task parameters in a system-resident table called the System Task Directory (STD). The recorded parameters include the name and size of the task, the disk address at which the task's image starts, and the address of the Partition Control Block (PCB) of the partition in which the task is to run.

1.5.1 Task States

An installed task is defined as a task that has an entry in the STD. It is neither resident in memory nor competing for system resources. The Executive considers it to be dormant until a running task or a command issued from a terminal requests the Executive to activate it. The Executive therefore recognizes two task states:

- Dormant. A dormant task is one that has been installed (has an entry in the STD), but has not been requested to run.

- Active. An active task is an installed task that has been requested to run. It remains active until it exits, or gets aborted. It then returns to the dormant state.

  An active task can be in one of two substates, ready-to-run or blocked.

  1. Ready-to-run. A ready-to-run task competes with other tasks for CPU time on the basis of priority. The ready-to-run task having the highest priority obtains CPU time and thus becomes the current task.

  2. Blocked. A blocked task is unable to compete for CPU time for synchronization reasons or because a needed resource is not available.

The distinction between dormant tasks and active tasks is important in a real-time system. A dormant task uses little memory; and yet when the task is needed to service a real-time event, the Executive can quickly and efficiently introduce it into active competition for system resources. An installed task's STD entry enables this quick response because it contains all the parameters the system needs to retrieve the requested task. Note that the number of installed, dormant tasks can, and usually will, far exceed the number of active tasks.

When the Executive receives a request to activate a dormant nonresident task, it performs a series of actions:

- It allocates the required memory resources on the basis of the task's priority.

- It brings the task into memory.

- It places the task in active competition for system resources with other resident tasks.

If tasks fully occupy the partition in which a task is installed and no resident task can be checkpointed (see Checkpointing in this chapter), the task is placed in a queue by priority with other activated tasks, each waiting for space to become available in its partition.

1.5.2 Fixed Tasks

A task can be loaded and locked into its partition. Such a task is called a fixed task. The Fix command (see the RSX-11M Operator's Procedures Manual) allows you to fix a task in memory. The Executive services subsequent requests to run the task more quickly because the task is resident in memory and does not have to be loaded from the disk before it can run. The system can fix a task in memory only when the partition in which it is to be fixed becomes available.

Fixed tasks remain physically in memory even after they finish execution. Therefore, the Executive does not have to reload them when they are again requested to run. However, tasks that can be fixed in memory must have re-entrant code if it is to be reused by another program or system user. Re-entrant code must be used because the task cannot be allowed to change its own internal data base if another program uses it. Only an Unfix or Remove command can free the memory that the task occupies.

NOTE

See the RSX-11M Operator's Procedures Manual for the Fix, Unfix, and Remove commands.

The following restrictions apply to tasks that you want to fix in memory:

- You must first install the task.
- You cannot fix an active task.
- You cannot fix a checkpointable task.

1.5.3 Priority

Active tasks compete for system resources on the basis of their relative priorities. The Executive gives control of the processor to the active task that has the highest priority and that also has access to all the other resources it needs. When this task becomes blocked (while waiting for I/O to complete, for example), the Executive looks for another task to use the processor. The chosen task is again the one that has the highest priority and has access to all the resources it needs.

You initially assign a default priority to a task when you task-build it. This priority is a number between 1 and 250 (decimal); higher numbers indicate higher priority. Later, you can change the priority when you install the task; or the system can change the priority while the task is running (see Swapping).

In an RSX-11M installation that mixes real-time applications with less urgent work, higher priority numbers should be assigned to the real-time tasks. This assignment ensures that the Executive gives processor time to the real-time tasks ahead of the others. Text editors are an example of real-time tasks, because they must respond within a short time period. Text editors, commonly used for program development or text processing, spend a large part of their time waiting for terminal I/O to complete and are therefore out of competition for processor time. However, when the I/O operation ends, the terminal needs a rapid response. To get the higher response, the installation system manager can assign to text editors a higher priority than that of more processor bound tasks like the Task Builder or Assembler.

1.5.3.1 **Establishing Task Priority** - You can establish task priority when you use the Task Builder to build a task from an object module. See the RSX-11M Task Builder Reference Manual for a description of the priority option.

1.5.3.2 **Installed Priority** - When you install a task using the Install command, you can specify a priority different from the one that you specified when you built the task. The priority specified in the Install command overrides the priority that was specified for the task in the Task Builder command. See the RSX-11M Operator's Procedures Manual for a complete description of the Install command.

1.5.3.3 **Altering Priority** - You may want to alter a task's priority after it is installed. The Alter command provides a way to change priority. With the Alter command you can change the task's static installed priority or change the task's running priority. However, you can make these changes only if the system supports the Alter Priority directive. See the RSX-11M Operator's Procedures Manual for a complete description of the Alter command.

1.5.4 Round-robin Scheduling

When numerous competing memory-resident tasks have equal priorities, the Executive tends to give processor time more often to those tasks that appear first in the System Task Directory (STD) queue. Entries with equal priorities normally appear in the STD in the order in which the tasks were installed. Therefore, the Executive favors tasks that were installed first. To avoid this problem, RSX-11M provides a system generation option called round-robin scheduling. Round-robin scheduling uses an algorithm that periodically rotates the execution of tasks of equal priority in the STD. The overall effect is that processor time is distributed more evenly among tasks. Each equal-priority task has its turn toward the head of the STD queue.

1.5.5 Checkpointing

In a programming system where many tasks of equal or different priorities are competing for memory space and system resources, the Executive must have a method of distributing processor usage and resources to all the tasks. The RSX-11M Executive uses a process called checkpointing to allocate system resources among tasks. The Executive uses task priority as the basis for the checkpointing scheme.

In some instances, an active task cannot get into memory and compete for processor resources because the partition in which it was installed is fully occupied. If the partition contains a task that has a lower priority and is checkpointable, the Executive moves that task out of memory and writes it on a disk to make room for the higher priority task. When the high priority task is finished, the Executive reloads the low priority task, which is now on the disk, to allow it to continue processing from the point at which it was interrupted. This roll-out, roll-in process is called checkpointing.

RSX-11M supports checkpointing in both user-controlled and system-controlled partitions. The objective is to avoid preempting a lower priority task, unless a higher priority task can be brought in to make use of the freed memory. This optimizes the use of system resources while maintaining a priority scheduling discipline.

1.5.5.1 **Disk Space for Checkpointing** - To checkpoint a task, checkpoint space equal to the size of the partition that contains the task must be available on disk. (Checkpoint space contains the checkpointed task while a higher priority task executes.) You can allocate checkpoint space either statically when building the task, or dynamically at run time. You can use both kinds of checkpointing to balance the advantages and disadvantages of the different allocation methods.

When you use the Task Builder to create a task from an object module, you can request checkpoint space allocation in the task image file on the disk; this is the same disk as the one on which the task resides. The task image file is the executable task on the disk. While the task is running, its checkpoint space is always allocated on disk, whether or not the Executive actually checkpoints the task.

You can use disk space more efficiently if you allocate checkpoint space dynamically. Instead of reserving disk space equal to the size of each checkpointable task, you can create one or more checkpoint files on disk to contain all checkpointed tasks. The size of the files depends on an estimation of the checkpoint space required at any given time. When the system allocates checkpoint space dynamically, tasks need not be built as checkpointable. Instead, you decide if a task can be checkpointed when the task is installed. You create a checkpoint file, independent of individual tasks, by issuing the ACS (Allocate Checkpoint Space) command from the terminal. Then, when the Executive needs to checkpoint a task, it writes the task out into the available space in the checkpoint file. A drawback to dynamic allocation of checkpoint space is that space in a checkpoint file may, at times, be filled. However, system performance may be improved if the checkpoint file is on a fast disk.

See the RSX-11M Operator's Procedures Manual for the Allocate Checkpoint Space (ACS) command.

1.5.6 Task Swapping

The Executive must deal with the situation that occurs when several active tasks with equal priorities compete for partition space in memory. A task cannot normally cause the Executive to checkpoint another task with the same priority. Therefore, a task of equal priority cannot get into memory. The Executive includes a task swapping algorithm that uses checkpointing to allow tasks of equal priority to successfully compete for memory.

Swapping is a variation of checkpointing that enables the Executive to checkpoint tasks with equal priorities in and out of memory. Swapping does not work, of course, unless the tasks are checkpointable. When an eligible task begins to run, the Executive adds a number to the task's normal running priority. This number is called the swapping priority and is used for swapping only. The old running priority still exists. As the task runs, the Executive decrements the swapping priority. Eventually, the sum of the decremented swapping priority and the task's running priority causes the running task to have a priority (for swapping) less than that of a competing task. When this occurs, the Executive checkpoints the running task to make room for the competing task. The Executive then places the checkpointed task in the queue of active tasks that are competing for memory. The swapping priority does not affect task scheduling or I/O dispatching, which are governed solely by the task's running priority.

1.5.7 The Shuffler Task

In trying to accommodate the execution of as many tasks as possible, the Executive moves tasks in and out of memory depending upon available space, priority, etc. This operation can result in fragmented memory, a situation in which many small tasks occupy memory with unused spaces in between. Taken individually, these spaces may not be large enough to allow large tasks to be loaded and executed. The Shuffler task, a system generation option, solves this problem by performing memory compaction in a system-controlled partition.

The Shuffler starts at the beginning of the system-controlled partition and tries to move (shuffle) all tasks that are sitting above a gap of free space down to the base of the free space. When possible, it also checkpoints any tasks that it encounters that are waiting for terminal input.

If there are some tasks still actively competing for memory in the partition, the Shuffler creates an ascending, priority-ordered list of the tasks in the partition. If the sum of the free space now in the partition and the space occupied by the low priority, checkpointable tasks in the partition is enough to allow the waiting task to run, the Shuffler checkpoints the lower priority tasks. The Shuffler then compacts memory again to make room for the waiting task.

The foregoing Shuffler action should result in all free space being at the top of the partition. However, there may be additional holes below tasks because some things (drivers and regions) cannot be shuffled. These additional holes cannot be reclaimed.

1.5.8 Extended Logical Address Space

An RSX-11M task specifies an address in a 16-bit word. The largest address that can be expressed in a 16-bit word is 65,536 bytes or 32,768 words (commonly referred to as 32K words). To avoid limiting the effective size of a task to only 32K words, a task can use overlays that you define when you use the Task Builder to build the task. Another option is that the task can use memory management directives to access greater amounts of memory.

1.5.8.1 **Overlays** - An overlaid task has parts called segments. The segments are the parts that overlay one another. The segments are also sometimes called overlays. The root segment, which is always in memory and never overlaid, and one or more overlay segments compose an overlaid task. The overlay segments can be read into memory as required. However, all the segments in memory at one time cannot exceed 32K words.

See the RSX-11M Task Builder Reference Manual for a complete description of overlay segments.

1.5.8.2 **Memory Management Directives** - Memory management directives allow task segments resident in memory to access more than 32K words of physical memory. The memory management directives, a subset of the Executive directives, use the KT11 hardware to map task addresses to different logical areas within the task. Instead of displacing task segments in memory, the task can reside entirely in memory and map its virtual addresses to different physical addresses.

RSX-11M defines three kinds of address space:

- Physical address space. Physical address space consists of the physical memory in which tasks reside and execute.
- Logical address space. Logical address space is the total amount of physical address space to which the task has access rights.
- Virtual address space. Virtual address space corresponds to the 32K of addresses that the task can explicitly specify in a 16-bit word. If a task does not use memory management directives, its logical and virtual address spaces directly correspond one to the other. However, if the task uses these directives, it can map its virtual addresses to different parts of its logical address space. The net effect is to allow a task's logical address space to exceed 32K.

The memory management directives also allow a task to expand dynamically its logical address space. In other words, a task can access logical areas that are not part of its static task image (the executable task produced by the Task Builder). A task can issue directives that create a new region of logical space and then map a range of virtual addresses to the newly created region. A task can also map its virtual addresses to logical areas that belong to another task. The mapped area then becomes part of the former task's logical address space.

The ability to create and map to a new region allows tasks to communicate with one another by means of shared regions. For example, at run time a task can create a new region of logical space, into which it writes a large amount of data. Any number of tasks can then access that data by mapping a range of their virtual addresses to the region. Another benefit of mapping to different regions is an ability to use a greater number of common routines. Tasks can map to the required routines at run time, rather than link to them when the tasks are built by the Task Builder.

1.6 RSX-11M INTERRUPT PROCESSING

The RSX-11M system recognizes two kinds of hardware interrupts: processor traps and external interrupts. Processor traps occur synchronously; that is, the same sequence of instructions causes the same processor trap to occur at the same place and time in the program. Processor traps usually have a cause originating from within the processor. See Processor Traps, in this chapter, for the causes of processor traps. External interrupts, which are usually caused by I/O devices, are asynchronous in that they may occur anywhere or at any time in the program's execution.

Programs that use input and output routines would spend most of their time waiting for I/O devices to complete their operations if it were not for the program interrupt facility of RSX-11M. The program interrupt facility allows asynchronous events, such as I/O completion, to interrupt the running program so that a routine can service the interrupting device. An interrupt is analagous to a subroutine jump. However, to preserve program integrity, interrupts are allowed to occur only after the completion of an instruction and before the start of the next instruction.

As an example of the program interrupt facility, programs can continue operation after starting a device, then allow the device to interrupt when it is ready to signal the program about its resulting status.

The addresses of the interrupt processing routines must be made known to the Executive. These addresses are called interrupt vectors and they are in the Executive's low memory area.

1.6.1 Interrupt Vectors

Each peripheral device controller in the RSX-11M system has a hardware pointer to its own pair of memory words. These words are located in the low memory of the Executive. One word contains a vector (address) for the device's interrupt service routine. The vector may be an entry point address or an Interupt Control Block address. If this vector is an entry point address, it becomes the contents of R7 the program counter or PC word) when the service routine begins its execution. For loadable drivers, the vector points to an Interrupt Control Block.

The other word is the processor status (PS) word. It contains the mode and the priority of operation for the interrupt routine. The hardware saves the status of the interrupted program (the PC and PS) before the interrupt routine begins its processing.

The Executive has an area called a stack in which it saves status, register contents, parameters, or any other data that it may need.

1.6.2 System Stack

RSX-11M maintains a push-down stack using general register 6, which is the stack pointer or SP. External interrupts, subroutine calls, and processor traps use this stack to save program status. When an interrupt occurs, the hardware first saves the current processor status word (PS) and the program counter (PC) on the stack. It then uses the new PS and PC from the trap and interrupt vector area in low memory, and begins processing the interrupt routine that handles that particular interrupt. A return from interrupt (RTI) instruction restores the original PS and PC values from the stack, thereby restoring the original interrupted program.

1.6.3 Processor Traps

A variety of errors and programming conditions cause the processor to trap to a set of fixed locations. These locations contain the PC and PS for the trap processing routines. Processor traps include the following:

- Power failure
- Odd addressing errors
- Stack errors
- Timeout errors
- Non-existent memory errors
- Memory parity errors
- Memory management violations
- Floating point processor exceptions
- Use of reserved instructions
- Use of the T-bit in the PS word
- Use of the IOT, EMT, and TRAP instructions

Processor traps cannot be masked off. That is, when they occur, the processor immediately enters the trap sequence of pushing the current PS and PC onto the current stack, retrieving the new PS and PC from a specific hardware trap vector, and executing the code that begins at the location specified by the trap vector.

Although there are several processor traps (see Interrupt and Trap Vectors, below), the trap of main interest is the emulator trap. The EMT instruction causes the emulator trap. This instruction calls the Executive whenever a user task has an Executive directive written into it that requests the Executive to perform some specific function (see Executive Directives below).

1.6.4 External Interruptions

External interrupts are hard-wired into one of four priority levels of the processor (labeled 4 to 7, with 7 being the highest priority). These interrupts are maskable in that they can cause an interrupt only if the priority level held in the processor status word is less than the priority of the interrupting source. When an interrupting device causes a new priority level to be loaded from its vector PS word, interrupts at the same or lower levels are blocked out. The system, however, remembers that the interrupts occurred and it processes them in turn by priority.

Certain traps, however, cannot be masked by the priority field in the PS word. These traps are: parity error, memory management violation, stack limit yellow, power failure (power down), and floating-point exception.

1.6.5 Interrupt And Trap Vector Locations

The following chart shows some of the interrupt and trap vectors used by RSX-11M interrupt and trap processing. The PC for the interrupt routine is taken from the specified memory location. The next word contains the new PS word.

Memory Location	Interrupt and Trap Vector
000	Reserved for DEC use
004	CPU errors
010	Illegal and reserved instructions
014	Breakpoint trap (BPT)
020	Input/output trap (IOT)
024	Power fail
030	Emulator trap
034	TRAP instruction
.	
.	
.	
244	Floating-point error
250	Memory management

For a complete list of vectors, see the pertinent PDP11 Processor Handbook.

1.6.6 System Traps

System traps are transfers of control (also called software interrupts) that provide tasks with another means of monitoring and reacting to events. The Executive initiates system traps when certain events occur. The trap transfers control to the task associated with the event and gives the task the opportunity to service the event by entering a user-written routine.

There are two distinct kinds of system traps:

- Synchronous System Traps (SSTs). SSTs detect events directly associated with program instruction execution. They are "synchronous" because they always occur at the same point in the program when previous instructions are repeated. For example, an illegal instruction causes an SST to occur.

- Asynchronous System Traps (ASTs). ASTs detect significant events that occur asynchronously to the task's execution; that is, the task has no direct control over the precise time that the event occurs. For example, the completion of an I/O transfer may cause an AST to occur.

To use the system traps, a task issues system directives that establish entry points for user-written service routines. Entry points for SSTs are specified in a single table. AST entry points are set by individual directives for each kind of AST. When a trap occurs, the task enters the appropriate routine via the specified entry point.

Debugging aid programs (On-line Debugging Tool and Executive Debugging Tool) can be entered from points, which are called breakpoints, that you insert into a memory-resident task. These breakpoints cause a breakpoint trap that transfers execution to the debugging aid program. The debugging aid, by means of its own table of trap vectors, can execute special processing for certain SSTs that can occur. The IAS/RSX-11 ODT Reference Manual discusses the On-line Debugging Tool in detail. The Executive Debugging Tool (XDT) is described in the RSX-11M Guide to Writing an I/O Driver.

1.7 EXECUTIVE DIRECTIVES

An Executive directive is a request from a task to the Executive to perform an indicated operation. A programmer uses Executive directives to control the execution and interaction of tasks.

Executive directives enable tasks to perform functions such as the following:

- Obtain task and system information
- Measure time intervals
- Perform I/O operations
- Manipulate a task's logical and virtual address space
- Suspend and resume execution of tasks
- Request the execution of another task
- Exit from a task

System directives allow tasks to exploit some major system functions, including the following:

- Event flags
- System traps
- Extended logical address space

RSX-11M MACRO programs execute Executive directives by using macro calls and the EMT 377 instruction. FORTRAN uses DIGITAL-supplied library routines to use the directives.

You should always use macro calls instead of directly executing the directive. Then, if system changes are made in the directive specifications, you need only to reassemble the program rather than edit the source code.

Listed below is a brief summary of the directive functions that are possible for RSX-11M. For a complete description of RSX-11M Executive directives, see the RSX-11M Executive Reference Manual.

Task Execution Control Directives

Abort Task	Causes the Executive to terminate the execution of the task named in this directive.
Cancel Time Based Initiation Requests	Causes the Executive to cancel all time-synchronized initiation requests for the execution of the task named in this directive, regardless of the source of each request.

Task Exit
Informs the Executive that the task issuing the Exit has completed its execution. Unless the exiting task is fixed, its memory is freed for use by other tasks.

Extend Task
Causes the Executive to modify the size of the task that issues this directive by a positive or negative number of 32-word blocks.

Request Task
Causes the Executive to request immediate execution of the task named in the directive.

Resume Task
Causes the Executive to resume the execution of a task that has issued a Suspend directive.

Run Task
Causes the Executive to schedule the execution of the task named in this directive at a time specified in terms of a time period from the issuance of the directive.

Suspend
Causes the Executive to suspend execution of the task that issued the Suspend until explicitly resumed, either by a Resume directive from another task or the MCR command, Resume.

Task Status Control Directives

Alter Priority
Causes the Executive to change the running priority of the installed and active task named in this directive.

Disable Checkpointing
Causes the Executive to make the task that issues this directive no longer checkpointable.

Enable Checkpointing
Causes the Executive to nullify the previously issued Disable Checkpointing directive.

Informational Directives

Get Partition Parameters
Causes the Executive to fill a 3-word buffer, which is specified in this directive, with parameters related to the memory partition specified in this directive or related to the task that issues this directive.

Get Region Parameters
Causes the Executive to fill a 3-word buffer, which is specified in this directive, with region parameters.

Get Sense Switches
Causes the Executive to return the settings of the 16 console switches to the task that issues this directive.

Get Task Parameters
Causes the Executive to fill a 16-word buffer with parameters related to the task that issues this directive.

Get Time Parameters
Causes the Executive to return the current time parameters (year, month, day, hour, minute, second, tick and ticks/second) of the task.

Event-associated Directives

Clear Event Flag
Causes the Executive to clear an event flag specified in the directive and return the previous polarity of the flag.

Cancel Mark Time Requests
Causes the Executive to cancel MARK TIME requests that have been made by the task that issues this directive.

Declare Significant Event
Causes the Executive to declare a significant event. The Executive scans the STD for the highest priority task capable of execution. It then saves the context of the currently executing task and starts the execution of the new highest priority task.

Exitif
Causes the Executive to cause an exit of the task that issues the directive if, and only if, a specified event flag is clear.

Mark Time
Causes the Executive to declare a significant event after the expiration of the time interval specified in the directive. If an event flag is specified in the directive, it is cleared when the directive is issued and set when the significant event occurs. If an Asynchronous System Trap (AST) entry point address is specified in the directive, an AST occurs at the time of the significant event.

Read All Event Flags
Instructs the Executive to return to the task that issued this directive the polarities of all 64 event flags in a 4-word buffer.

Set Event Flag
Causes the Executive to set an indicated event flag and return the previous polarity of the indicated flag (without a declaration of a significant event).

Wait For Significant Event
Causes the Executive to suspend the execution of the task that issues the directive until the next significant event occurs.

Wait For Logical Or Of Event Flags

Causes the Executive to suspend the execution of the task that issues the directive until one or more specified event flags of a group of event flags is set.

Wait For Single Event Flag

Instructs the executive to suspend the execution of the task that issues the directive until an event flag that is specified in the directive is set.

Trap-associated Directives

AST Service Exit

Causes the Executive to terminate the execution of the AST service routine.

Disable AST Recognition

Causes the Executive to disable AST recognition for the task that issues this directive. The ASTs are queued and only their recognition is inhibited.

Enable AST Recognition

Causes the Executive to enable AST recognition for the task that issues this directive.

Specify FPP Exception AST

Informs the Executive that the specified AST routine within the task is to begin execution whenever a floating-point processor exception occurs, or that floating-point processor exception ASTs are no longer wanted.

Specify Power Recovery AST

Informs the Executive whether or not power recovery ASTs are wanted for the task that issues this directive. If the ASTs are wanted, this directive indicates where control is to be transferred when the AST occurs.

Specify Receive Data AST

Informs the Executive whether or not receive data ASTs for the task issuing this directive are wanted. If the ASTs are wanted, task execution is transferred to the address of the AST service routine within the task when data is placed in the task's receive queue.

Specify Receive By Reference AST

Informs the Executive to transfer control to an address in the task specified in the directive when the Receive-by-Reference AST occurs, or that receive-by-reference ASTs are no longer desired for the task that issued this directive.

Specify SST Vector Table For Debugging Aid

Specifies the address of a table of synchronous system trap service routine entry points for use by ODT or other debugging aids.

Specify SST Vector Table For Task

Informs the Executive that the task that issues this directive contains a table of addresses of service routines to be executed upon task trap or fault conditions.

I/O and Intertask Related Directives

Assign LUN

Causes the Executive to assign a physical device unit to a logical unit number (LUN). The LUN, device name, and device unit number are specified in this directive.

Connect To Interrupt Vector

Causes the Executive to allow a task to process hardware interrupts by a routine specified in the directive. The Interrupt Service Routine (ISR) must be in the task's own space.

Get LUN Information

Causes the Executive to return information regarding the logical unit specified in the directive to the task that issues this directive.

Get MCR Command Line

Causes the Executive to transfer an MCR (terminal) command line to the task that issues this directive.

Queue I/O Request

Causes the Executive to queue an I/O request for the issuing task. This request is queued by priority for a logical unit which is assigned to a physical unit. An event flag, an AST, and an I/O status block may be specified as I/O completion indications.

Queue I/O Request And Wait

Similar to the queue I/O request directive except for one aspect. The Queue I/O Request And Wait directive specifies an event flag and the Executive executes an implied Wait For Single Event Flag directive.

Receive Data

Informs the Executive that the task that issues this directive is ready to receive data (in a 13-word data block) that has been sent from another task by means of the Send directive.

Receive Data Or Exit

Causes the Executive to attempt to receive data (dequeue a 13-word data block) for the task that issues this directive. If no data is received, the task that issues this directive exits.

Send Data

Causes the Executive to declare a significant event and to queue the 13-word block of data that the task named in this directive is to receive.

Memory Management Directives

Attach Region — Causes the Executive to attach the task that issues this directive to a static common region or to a named dynamic region.

Create Address Window — Causes the Executive to create a new virtual address window by allocating a window block from the header of the task that issues this directive and establishing the window's virtual address base and size.

Create Region — Causes the Executive to create a dynamic region in a system-controlled partition and, as an option, attach it to the task that issues this directive.

Detach Region — Causes the Executive to detach the task that issues this directive from the previously attached region that is specified in this directive.

Eliminate Address Window — Causes the Executive to delete an existing address window, unmapping it first if necessary.

Get Mapping Context — Causes the Executive to return a description of the current window-to-region mapping assignments.

Map Address Window — Causes the Executive to map an existing window to an attached region.

Receive By Reference — Causes the Executive to dequeue the next packet in the receive-by-reference queue of the task that issues this directive.

Send By Reference — Causes the Executive to insert a packet containing a reference to a region into the receive-by-reference queue of a receiver task that is specified in this directive.

Unmap Address Window — Causes the Executive to unmap the window that is specified in this directive.

1.7.1 Event Flags

The execution of certain directives causes significant events to occur. In fact, most significant events are caused, either directly or indirectly, by system directives.

A significant event occurs when a task issues a system directive that implicitly or explicitly suspends a task's execution, or when an external interrupt occurs that can affect a task's execution. Event flags are associated with significant events. When a significant event occurs, the event flag indicates the specific cause of the significant event.

The Executive uses significant events and event flags to manage task execution. However, tasks can also use significant events to coordinate internal task activity and to communicate with other tasks. For example, a task can issue an Executive directive to associate an event flag with a specific significant event. When that event occurs, the Executive sets the associated flag. Therefore, by testing the state of the flag, a task can determine whether or not the event has occurred.

Sixty-four event flags are available to enable tasks to distinguish one event from another. Each event flag has a corresponding event flag number. The first 32 flags are local to each task and are set or cleared as a result of each task's requirements. The second 32 flags are common to all tasks and are therefore called global or common event flags. Global flags can be set or cleared as a result of any task's operation. Tasks use global flags to communicate with other tasks because one task cannot refer to another task's local flags. Eight of the local event flags and eight of the common event flags are reserved exclusively for the Executive.

1.8 THE MCR INTERFACE

You communicate with RSX-11M by entering commands at a terminal. The terminal driver directs the commands to the Monitor Console Routine (MCR) processor. The MCR processor either executes the commands itself, or it activates a system or user-written task that can service the commands.

MCR commands allow you to:

- Start up the system
- Manage peripheral devices
- Control task execution
- Obtain system and task information
- Activate system or user-written tasks that request input from the terminal

The MCR commands that control task execution are particularly significant to system performance. You must use an MCR command (Install) to install a task into the system. Therefore, you establish the base of installed tasks, which the Executive, other installed and active tasks, and further MCR commands can manipulate.

1.8.1 Privileged Commands

To restrict the use of commands that directly affect system performance, RSX-11M considers some MCR commands and command options to be privileged. You can issue a privileged command only from a privileged terminal. In multiuser protection systems, individual users are either privileged or nonprivileged; when a user logs on, the terminal assumes the privilege status assigned to that user's identification code (UIC). A user can issue an MCR command at a privileged terminal to modify the privilege status of any other terminal connected to the system. If multiuser protection support is not included during system generation, all terminals are privileged.

1.8.2 External Scheduling Of Task Execution

An important MCR function is the external scheduling of task execution. This type of scheduling works in conjunction with the Executive's priority driven internal scheduling of active tasks. You can include time parameters with the command that activates an installed task. The time parameters request the Executive to run a task:

- At a specified time from the current moment
- At a specified time from clock unit synchronization
- At an absolute time of day
- Immediately

All of these time options are available with or without periodic rescheduling. RSX-11M also supports an unlimited number of programmed timers for each task in the system. The user task can create its own timer, which the Executive then decrements at regular intervals. When the timer reaches zero, the Executive sets an event flag or generates an Asynchronous System Trap (AST) that passes control to the task at a prespecified address.

1.9 TERMINAL OPERATION

In RSX-11M, a variable number of terminals can operate concurrently. In addition, each terminal operates independently of others in the system to allow each to run a different task. In a system that supports multiuser protection, a user must log onto a terminal before issuing further commands. In other RSX-11M systems, a user can issue commands whenever the terminal displays an appropriate prompt.

1.9.1 Attached Terminals

RSX-11M allows tasks to request input from a terminal. To ensure that a requesting task receives input intended for it, the task usually attaches to the terminal. While the task is attached, the terminal directs all input to the attached task, with one exception. The exception is a control C character (the C key pressed while pressing the CTRL key), which gains the attention of the MCR processor. An attached terminal ensures that a soliciting task properly receives its input; but it also allows a user to interrupt the task's control of the terminal to communicate with MCR. Note that attaching to the terminal is a function of the task rather than of a user.

Some applications may require that a user be denied access to MCR but allowed access to a specific task only. In this case, a task can attach to the terminal with a special subfunction. The subfunction causes the system to generate an AST for the attached task whenever someone enters unrequested input, including CTRL/C, at the terminal. However, making the terminal a slave terminal is another way of doing this.

1.9.2 Slave Terminals

When your installation needs to dedicate a terminal exclusively to one or more tasks, you issue an MCR command (or a task issues a special I/O function) that sets the terminal to slave status. The difference between a slave terminal and an attached terminal is that the system ignores all unsolicited input, including CTRL/C, that is entered at a slave terminal. Until you issue another MCR command to delete the slave status, the terminal can only be used to communicate with the task soliciting input from the terminal. An I/O function issued by a task can also delete the slave status of the terminal. Slave terminals are often dedicated to real-time applications.

1.10 MULTIUSER PROTECTION

Multiuser protection, a system generation option, allows an RSX-11M installation to monitor and control individual users of the system. Individual users are either privileged or nonprivileged. The system manager, who is the one assigned responsibility for system configuration and operation, assigns a user identification code (UIC) to each user, which determines the user's privilege status. When logging onto a terminal, the user supplies a last name or UIC and a password. If the user gives a name, the system finds the associated UIC. The system then checks that the password matches the last name or UIC, and sets the terminal to privileged or nonprivileged status, according to the user's UIC.

1.10.1 Public And Private Devices

In a multiuser protection system, some commands allow you to do things that are not allowed in systems without multiuser protection. For example, the Allocate command allows you (or any user) to allocate a device (a disk drive) as your private device; allocating the device prevents other nonprivileged users from accessing it.

A nonprivileged user can access a private device that he has allocated to perform MCR functions that are normally privileged. These functions include preparing a disk or magnetic tape for use by the RSX-11M file system.

To complement the private device feature, multiuser protection allows the system manager or privileged user to declare certain devices to be public. Public devices cannot be allocated to individual users. By declaring a line printer to be public, for example, the system manager can ensure that all users have access to that commonly used output device.

1.11 SYSTEM MAINTENANCE

1.11.1 Error Logging

RSX-11M provides an error logging facility as a system generation option for systems that are 24K words or larger. The error logging facility monitors the hardware reliability of an RSX-11M system; it continually detects and records information about disk, DECtape, magtape, and memory errors as they occur, regardless of whether or not the error is recoverable. The Executive automatically retries recoverable errors. However, you might be unaware that the error occurred. Therefore, at user-determined intervals, a formatting task can be run to generate individual error and summary reports on some or all of these errors.

Please note that only the following four types of errors are loggable:

- Device errors (disks, magtapes, DECtapes)
- Undefined interrupts
- Timeout
- Memory parity errors

In summary, the error logging facility performs the following functions:

- Detects a hardware error as it occurs (done by Executive modules)
- Gathers information about the error
- Stores the information in a file
- Formats the information to produce an error report

Control of the facility is shared between routines in the Executive and specific error logging tasks. These routines and tasks interface with each other to carry out the four operations described above.

You can generate a wide variety of error reports. Among many options, you can specify a report that covers only a certain time period, a certain device or group of devices, or perhaps a certain type of error. You can also request a report that contains only information on individual errors, one that contains only summary information, or one that contains both kinds of statistics.

Because the error log files may be written to a removable volume, an operator can generate the reports either on site or at any other RSX-11M installation that supports the error logging facility.

1.11.2 Diagnostic Tasks

RSX-11M also provides a group of diagnostic tasks which you can incorporate into the Executive support at system generation time. A diagnostic task tests a specific device to identify the source of any errors. RSX-11M diagnostic tasks test for malfunctions on most disks, DECtapes, magnetic tapes, and terminals. The tasks are simple to use and require little memory space.

When used in connection with error logging reports, the diagnostic tasks can significantly reduce system downtime. The system manager should regularly generate error reports to check on hardware performance. When a number of errors indicates that a particular device is beginning to malfunction, the manager can run the diagnostic task for the erring device to help isolate the source of the errors.

Each diagnostic task has two modes of operation: customer mode and service mode. In customer mode, the user activates the appropriate task, which then runs to completion and reports its findings. (Because the tests destroy any data resident on the device being tested, only authorized users should be allowed to run diagnostic tasks.) Service mode is intended for use by DIGITAL Field Service engineers. Service mode allows the user to modify the test content initially and to interrupt the running test to make further modifications.

1.11.3 Power Failure Restart

RSX-11M can execute a power failure restart that smooths out intermittent short-term power fluctuations with little loss of service or data. Power failure restart functions in four phases:

- When power begins to fail, the CPU traps to the Executive, which stores volatile register contents, thereby bringing system operations to a controlled halt.
- When power is restored, the Executive again receives control and restores the preserved state of the system.
- The Executive then schedules all device drivers that were active at the time of the power failure at their power-fail entry points. Drivers have the option of being scheduled one of two ways:

  1. Whenever power fails
  2. Only when power fails while the driver is servicing an I/O request

  The drivers can then make any necessary restorations of state (repeat an I/O transfer, for example).
- The Executive then determines if any user-level tasks have requested notification of power failure by issuing a system directive requesting an AST on power recovery. The Executive initiates ASTs for any tasks that have requested them.

CHAPTER 2

MEMORY RESOURCE ALLOCATION

2.1 INTRODUCTION

Chapter 2 contains information about how the Executive manages, structures, allocates, and deallocates memory resources in RSX-11M. Any discussion of memory functions in RSX-11M necessarily overlaps the closely related functions of task management and processing. However, this discussion emphasizes memory allocation, deallocation, and management to allow a more logical and coherent presentation of memory in RSX-11M.

The functions of the core allocation routines, the Shuffler and the Loader, are part of Executive memory management. However, the term "memory management" also refers to the KT11 Memory Management Unit, which is hardware and not software. The use of the term "memory management" has been avoided in this manual where confusion between the Executive's role and the hardware's role in memory management would arise. At the end of this chapter, flow diagrams show important processes that the Executive performs to allocate and manage memory.

2.1.1 Memory Addressing

Because of the 16-bit word size of the PDP-11, an RSX-11M task can have an address no larger than 177777(8) (an addressing range of 32K words for nonprivileged tasks). In RSX-11M, you can use a task that contains overlays to avoid limiting its size to its addressing range. An overlaid task contains segments -- a root segment that is always in memory, and any number of other segments that are loaded into memory when required. When task segments are not in memory, they reside on disk. Large task segments that are in memory may not be able to access large amounts of disk-based data because the data may not fit into the available memory with the task. A heavily overlaid task that transfers large amounts of data to another task via disk incurs a throughput penalty because of the many I/O transfers needed to move segments in addition to those I/O transfers needed by the task's function.

The combined size of an overlaid task's segments may exceed 32K, which is the limit imposed by 16-bit addressing. Normally, the sum of task segment sizes in memory is 32K or less. However, a non-privileged task can exceed the 32K physical size imposed by the 16-bit address structure by using the memory management programmed logical address space (PLAS) directives. Combining the PLAS directives with memory-resident overlays is an effective way to avoid throughput problems caused by many I/O transfers. With this combination, I/O transfers occur to move only data to another task and, because the entire task is in memory, all or most of the task segments do not have to be loaded or unloaded during task execution.

Task throughput can be faster if all or a greater portion of the task is resident in memory during task execution. RSX-11M contains a group of memory management directives that provide the task with this capability. The directives overcome the 32K word addressing restriction. They allow the task to change the physical memory locations referred to by a given range of addresses. Using these directives, a task can increase its execution speed by reducing its disk I/O requirements at the expense of increased memory requirements.

The memory management directives that a task can use for expanding the 32K range of accessible addresses are:

CRRG$ -- Create Region

ATRG$ -- Attach Region

DTRG$ -- Detach Region

CRAW$ -- Create Address Window

ELAW$ -- Eliminate Address Window

MAP$ -- Map Address Window

UNMAP$ -- Unmap Address Window

SREF$ -- Send by Reference

RREF$ -- Receive by Reference

GMCX$ -- Get Mapping Context

GREG$ -- Get Region Parameters

The use of these directives is fully described in the RSX-11M Executive Reference Manual. The RSX-11M Task Builder Reference Manual describes overlay structures and overlaid tasks.

2.1.2 Memory Management - An Overview

In a mapped system, the KT11 Memory Management Unit associates task addresses with available physical memory. This process, which is transparent to the user, is called mapping. Addresses used within a task are virtual addresses and their correspondence to actual physical memory addresses is known to the KT11 unit only. However, memory management directives can control and manipulate the KT11, which physically performs the address mapping.

A privileged task can address all of available memory by directly using the KT11 Memory Management Unit. There is some danger in doing this because the programmer must be very certain that the task does not corrupt system space, system routines and data (for instance, the Executive itself, its pool space, or the I/O page), or other tasks.

2.1.3 Virtual And Logical Addresses

Virtual and logical addresses, and virtual and logical address space are concepts that provide a basis for understanding the functions performed by memory management directives and the use of task windows.

- Physical addresses - Memory is divided into discrete addressable parts called bytes. They are numbered according to their position in memory. Therefore, the lowest byte is 0 and the highest byte is whatever the upper limit of memory may be for a particular system; for example, 32K, 64K, etc. The assigned number is called the physical address.

  A task contains virtual addresses (for example, 0 through 2200). The Task Builder relocates the task's virtual addresses in an unmapped system by a number represented by the base address of the partition in which it is installed. After installation, the task's addresses refer to logical addresses of memory, which always correspond to the same physical memory in an unmapped system (unless you change the partition or task code). Therefore, the addresses have an actual one-to-one relationship to physical memory. The same relationship exists any time the task is in memory. The logical addresses may not be from 0 through 2200. For example, after the task is installed in the partition, the task's virtual address 0 may become logical and physical address 17000 because the Task Builder added in the offset, which is equal to the partition base address. In a mapped system, the virtual addresses remain the same but the logical addresses may change due to Executive processes (checkpointing, swapping, etc.). Therefore, the logical addresses do not always refer to the same physical memory. If the task uses memory management directives, the logical addressing can be changed by the task to include any part of physical memory that it is allowed to access.

- Virtual addresses -- A task's virtual addresses are the addresses within the task. The PDP-11's 16-bit word length (a mapped system) imposes the address range of 32K-words on the virtual addresses. Therefore, these task addresses could include addresses zero through 177777(8) depending on the length of the task. However, in a system that uses the KT11 (mapped system), these task addresses may not be the same as the actual addresses of the memory in which the task resides. The KT11 Memory Management Unit maps the task's virtual addresses to the logical addresses of memory.

- Virtual address space -- A task's virtual address space is that space encompassed by the range of virtual addresses that the task uses. With the CRAW$ memory management directive, a task can divide its virtual address space into segments called virtual address windows. By using address windows, you can manipulate the mapping of virtual addresses to different areas of physical memory (see Virtual Address Windows below).

- Logical addresses -- A task's logical addresses are the actual physical memory addresses that the task can access.

- Logical address space -- The task's logical address space is the total amount of physical memory to which the task has access rights. The physical memory represented by the logical addresses may or may not be continuous. In other words, though a task's virtual addresses may be continuous, its logical address space may be divided among non-adjacent parts of physical memory. Using the CRRG$ (Create Region) memory management directive, you can divide the task's logical address space into various areas called regions. Each region is a continuous block of memory; however, the regions may not be adjacent.

If the capabilities of the memory management directives were not available, a nonprivileged task's virtual address space and logical address space would directly correspond. That is, a single virtual address would always point to the same logical location. Both types of address space would have a maximum size of 32K. However, you can use directives to assign a range of virtual addresses (a window) to different logical areas (regions), thereby extending a task's logical address space beyond 32K words.

Figure 2-1 shows a virtual address in a user task translated into a logical address in physical memory by the KT11 Memory Management Unit. The paragraphs following Figure 2-1 briefly describe the task virtual space, memory management, and task logical space relationships. A complete discussion of tasks, task windows, and regions can be found in the RSX-11M Task Builder Manual. Task mapping is also discussed in Chapter 4, Privileged Tasks.

2.1.4 Task Windows

Referring to Figure 2-1, which illustrates a mapped system, you can observe that a large 32K user task contains three distinct areas of continuous space called "windows". When referring to task windows (a file window is a similar but slightly different concept) the term, window, is a construct that encompasses and defines an area of continuous, virtual, program space in the task. Windows must have a specified size and starting address. The window size can be from 32 words to 32K-32 words and windows must start on a 4K address boundary. Figure 2-1 shows three windows that are not continuous in the task's virtual address space. However, the space within each window is continuous. In this task, the size of window 0 is 11K; the size of window 1 is 11K; and the size of window 2 is 8K. The concept of windows exists for the following specific reason.

By using the concept of windows and the Memory Management directives, a nonprivileged task can access a larger logical memory space than that implied by the 32K virtual addressing range and normally accessible by the 16-bit address. A task can, in fact, only access 32K of memory at one time. However, a nonprivileged task can change its access to logical addresses (real, physical memory). The area that your program accesses can be changed by the program during program execution. The process of accessing different logical areas of memory is called "mapping". By referring to Figure 2-1, you can see that Window 1 in the task is mapped to Static Common Region 1 in physical memory. The Window 1 mapping can be changed by the task to map to Static Common Region 0 in physical memory. In effect then, though a task is limited to a range of 32K virtual addresses, a task can access all the physical memory available to it (determined by the way that you set up the mapping) by changing the mapping of its windows to different logical addresses. Figure 2-1 provides a visual description of the concept of mapping to different logical addresses.

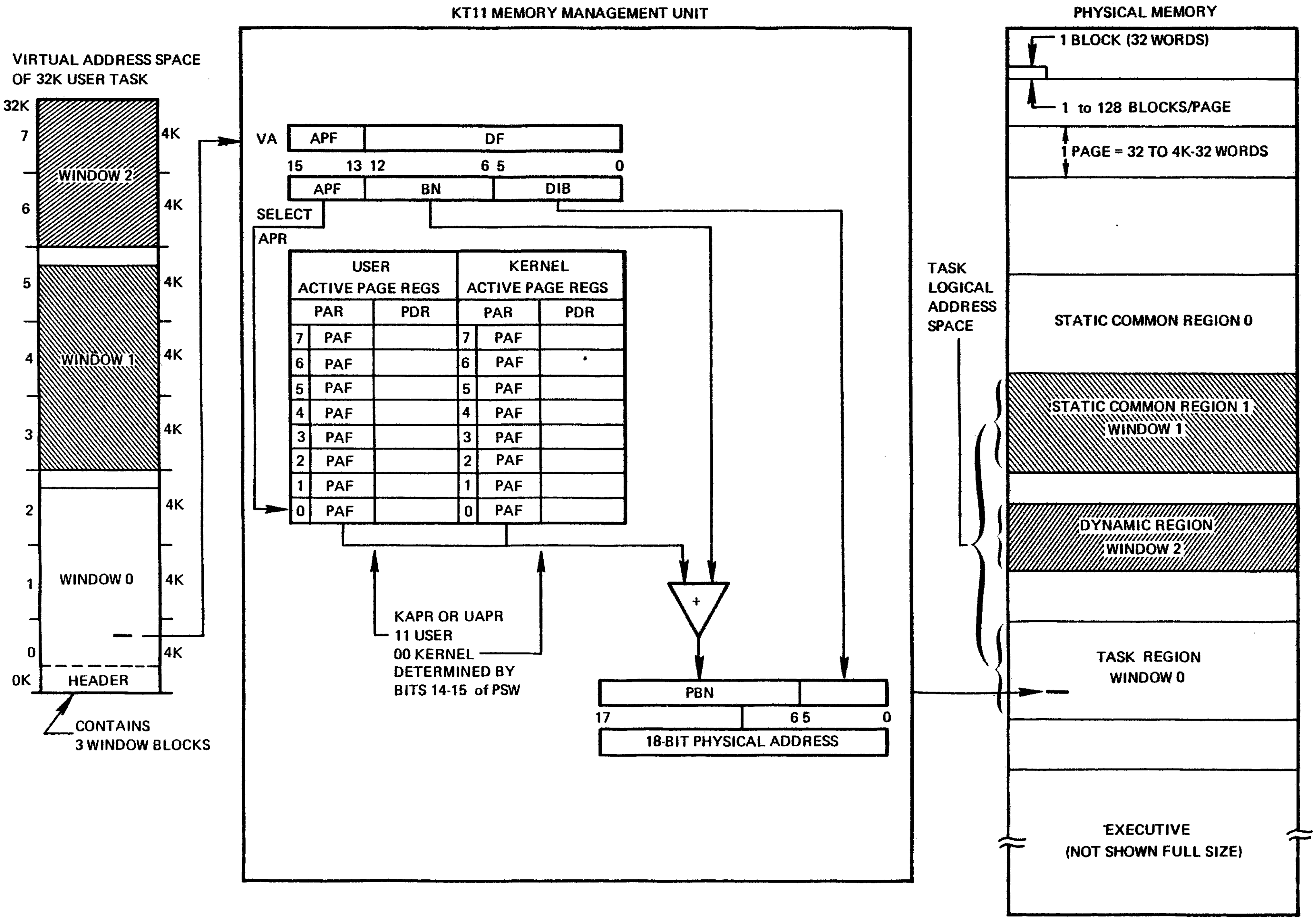


Figure 2-1 Memory Management - Virtual to Logical Address Space Relationship

Figure 2-1 shows a task that has three windows. One of the windows can map to two different logical areas of memory. Window 0 in the task maps to the task region. The task region is 11K in size and this size corresponds to the size of Window 0 in the task. The task region contains the task root and header. The program cannot change this mapping because window 0 is a default of the Task Builder and must contain the root and header. If no windows were specified, the Task Builder would create Window 0 for the entire task to map to its own logical space in memory. Window 1 maps to Static Common Region 1. Window 1 and Static Common Region 1 are both 11K in size. The task can change the mapping of Window 1 to map to Static Common Region 0. Observe here that the task can access 11K more physical memory than it occupies. In other words, a task of this size (32K), without windows, can access 32K of memory. The task that is illustrated can access 43K of memory.

The last window, Window 2, accesses the Dynamic Common Region. Both Window 2 and the Dynamic Common Region are 8k in size.

Note that the spaces that exist between the windows in the illustrated task do not refer to any logical memory address because no window (mapping) exists for those spaces.

The discussion now proceeds to setting up the task's windows. This is done by defining task window blocks to the Task Builder.

To manipulate virtual address mapping to various logical areas, the programmer must first divide a task's 32K of virtual address space into segments. These segments are task (virtual address) windows. Each window encompasses a continuous range of virtual addresses. The first address of the window address range must be a multiple of 4K (the first address must begin on a 4K boundary) because of the way that the KT-11 Memory Management Unit uses Aetire Page Registers (APRs). The number of windows defined in a task can vary from 1 to 7. Window 0 is not available to non-privileged tasks. The size of each window can range from a minimum of 32 words to a maximum of 32K minus 32 words.

A task that includes directives that dynamically manipulate address windows must have task window blocks set up in its task header as well as Window Definition Blocks in the code for use by the Create Address Window directive. The Executive uses task window blocks to identify and describe each currently existing window. When linking the task, the programmer specifies the number of window blocks to be set up by the Task Builder. The number of blocks should equal the maximum number of windows that will exist concurrently while the task is running.

A window's identification is a number from 0 to 7, which is an index to the window's corresponding window block. The address window identified by 0 is the window that always maps the task's header and root segment. The Task Builder creates window 0, which the Executive uses to map the task. No directive should specify window 0.

When a task uses memory management directives, the Executive views the relationship between the task's virtual and logical address space in terms of windows and regions. Unless a virtual address is part of an existing address window, the address does not point anywhere. This is a point to watch when setting up windows with the Create Address Window directive (CRAW$). Similarly, a window can be mapped only to an area that is all or part of an existing region within the task's logical address space.

Once a task has defined the necessary windows and regions, the task can issue memory management directives to perform operations such as the following:

- Map a window to all or part of a region.
- Unmap a window from one region in order to map it to another region.
- Unmap a window from one part of a region in order to map it to another part of the same region.

NOTE

> It is currently possible for a task with outstanding I/O to unmap from a region although it cannot detach from a region under this condition. Because this feature may be impossible to support in future releases of the system, DIGITAL recommends that users consider carefully before designing an application based on this ability.

2.1.4.1 **Task Window Block** - A task that includes Memory Management directives to manipulate address windows must have window blocks set up in the task header. The Executive uses the task window blocks to identify and describe each currently existing window. When linking the task you must specify the number of window blocks to be set up by the Task Builder. To do this, you specify a number in the WNDWS options of the Task Builder to define the number of windows to be used (in addition to the default window, Window 0). The Task Builder reserves space in the task header for window blocks - one block for each window. The Task Builder always reserves at least one block (for Window 0) as a default.

The label block group in the task header contains the number of task windows. The word where this number is found is L$BWND. The number that is contained here does not include windows for libraries.

The variable part of the task header contains the window blocks. You must specify window size, address limits, etc. (see Window Definition Block). However, task window blocks are filled in by the Executive when it obtains the information that you have described in the Window Definition Block (WDB) and Region Definition Block (RDB). Note that the RSX-11M Executive Reference Manual contains all the information that you need to establish address windows, regions, and their respective blocks in your code. Also, the RSX-11M Task Builder Manual contains a description of the task header.

2.1.4.2 **Window Definition Block (WDB)** - The Window Definition Block is a coding structure that you must create by using the WDBDF$ and WDBBK$ macros if you are using MACRO-11. You can create this block with an 8-word integer array if you are using FORTRAN. The Window Definition Block defines all the parameters that your code, the

Executive, and the Task Builder need to use windows. The Window Definition Block contains the:

- Window's base APR:
  APRO = Virtual base address 0
  APR1 = Virtual base address 4K
  APR2 = Virtual base address 8K
  .
  .
  .
  APR7 = Virtual base address 28K
- Size of the window in 32-word blocks
- Region ID of the region to map
- Offset within region to be mapped in 32-word blocks
- Window status word bit definitions
- Send/Receive buffer virtual address

The window's base APR and the size of the window are information that the Executive needs. The region ID relates the window to the Region Definition Block (RDB) which is also created by you. The offset within the region determines where the mapped area starts within the region. You can change the offset to move the area that is mapped within the region. The RSX-11M Executive Reference Manual contains a complete description of creating and using the WDB and the RDB.

2.1.5 Regions

A task's current window-to-region mapping context determines the part of the task's logical address space that the task can access at one time. A task's logical address space can consist of various types of region:

- Task Region -- The task region is a continuous block of memory in which the task runs.
- Static Common Region -- A static common region is an area defined by an operator at run time or during system generation; for example, a global common area.
- Dynamic Region -- A dynamic region is a region created dynamically at run time by using the Create Region (CRRG$) memory management directive in the task.

Tasks refer to a region by using a region ID, which the Executive returns to the task after the task creates the region. The Executive returns the region ID in the R.GID field of the Region Definition Block that the user must create before using the Create Region directive. Region ID 0 always refers to a task's task region. All other region IDs are actually addresses of the attachment descriptor maintained by the Executive in the system's dynamic storage region.

Figure 2-1 shows a sample collection of regions that could make up a task's logical address space. A task's logical address space can expand and contract dynamically as the task issues the appropriate memory management directives. The header and root segment are always part of the task region. Therefore, the task header and root segment always use window 0 (UAPR 0) and region 0. Because a region occupies a continuous area of memory, each region is shown as a separate block.

2.1.5.1 **Shared Regions** - Address mapping not only extends a task's logical address space beyond 32K words, it also allows the space to extend to regions that have not been linked to the task at task-build time. One result is an increased potential for task interaction by means of shared regions. For example, a task can create a dynamic region to accomodate large amounts of data. Any number of tasks can then access that data by mapping to the region. Another result is the ability of tasks to use a greater number of common routines. Tasks can map to required routines at run time, rather than link to them at task-build time.

2.1.5.2 **Attaching to Regions** - A task attaches to a region to make that region a part of the task's logical address space. A task can map only to a region that is part of the task's logical address space. There are three ways to attach a task to a region:

1. All regions that are linked to a task at task-build time are automatically attached.
2. A task can issue a directive to attach itself to a named static common region or a named dynamic region.
3. A task can request the Executive to attach any region within its own logical address space (other than its task region) to another specified task.

Attaching identifies a task as a user of a region, and prevents the system from deleting a region until all tasks have been detached from it. (It should be noted that fixed tasks do not automatically become detached from regions upon exiting.)

2.1.5.3 **Region Protection** - A task cannot indiscriminately attach to any region. The following criteria determine how tasks can attach to regions outside their logical address space:

- Each region has a protection mask to prevent unauthorized access. The mask indicates the types of access (read, write, extend, delete) allowed for each category of user (system, owner, group, world). The Executive checks that the requesting task's User Identification Code (UIC) allows it to make the attempted access. The attempt fails if the protection mask denies that task the access it wants.
- When a task creates a dynamic region, it may or may not give that region a name. If the dynamic region is named, any task can map to it as long as it knows the name and there is no protection violation. If a dynamic region is unnamed, a task can map to the region only if the task that created the dynamic region issues a Send By Reference directive addressed to the requesting task.
- Any task can issue a Send By Reference directive to attach any region (except the task region) to another specific task. The reference sent includes the access rights with which the receiving task attaches to the region. The sending task can only grant access rights that it has itself.
- Any task can map to a named static common region as long as there is no protection violation.

2.1.5.4 Region Definition Block (RDB) - You must create the Region Definition Block for each dynamically created region with the RDBDF$ and RDBBK$ macros if you are using MACRO-11. You can create the RDB with an 8-word single-precision array if you are using FORTRAN.

The RDB contains the:

- Region ID
- Region size in 32-word blocks
- Region name (in RAD50)
- Name of the partition (in RAD50) in which to create the region
- Region status word
- Region default protection

The RSX-11M Executive Reference Manual contains a complete description of creating and using the RDB.

2.2 MEMORY ALLOCATION

This section contains a textual discussion of the major functions and units of memory allocation. These are: checkpointing, swapping, memory compaction (the Shuffler), and loading (the Loader task). Flow diagrams of the major routines that are involved in these processes are included at the end of the memory allocation section.

This section also includes flow diagrams of the Loader, Shuffler, $NXTSK, and related routines called by $NXTSK.

2.2.1 Checkpointing

RSX-11M supports checkpointing in both user- and system-controlled partitions. The objective of checkpointing is to prevent lower priority tasks from using main memory and thereby preventing its use by higher priority tasks.

2.2.1.1 Checkpointing in User-controlled Partitions - Checkpointing in a user-controlled partition occurs under one of two conditions:

- A task requires the user-controlled main partition and has a higher priority than any other task currently occupying it or any of its subpartitions. Furthermore, all the occupying tasks must be checkpointable and have checkpointing enabled. If all of these conditions are met, the Executive checkpoints all the tasks that occupy the partition and gives control of the partition to the higher priority task.
- A task requires a subpartition of the user-controlled main partition and a lower priority task occupies the main partition or the subpartition into which the task is to be loaded. Furthermore, the occupying task must be checkpointable and have checkpointing enabled. If all these conditions are met, the Executive checkpoints the task that occupies the partition and gives control of the subpartition to the higher priority task.

2.2.1.2 Checkpointing in System-controlled Partitions - Checkpointing in a system-controlled partition occurs as the result of a memory allocation failure. That is, the Executive tries to allocate a continuous section of a system-controlled partition to a task and it cannot find an unoccupied memory area of sufficient size. In this case, the Executive re-examines the list of allocated areas in the partition to determine whether it can form a free space of sufficient size by checkpointing one or more neighboring tasks. As with user-controlled partitions, each task considered for checkpointing must be of a lower priority, it must be checkpointable, and it must have checkpointing enabled.

The Executive scans the list of allocated areas in the partition looking for a series of neighboring tasks, possibly separated by gaps of free space, where each task satisfies the checkpoint criteria. If the sum of the memory occupied by such a series of tasks and gaps satisfies the memory requirement for the higher priority task, the tasks are checkpointed and the higher priority task is allocated the released memory. If such a series of neighboring tasks cannot be found and memory compaction was generated for the system, the Executive calls the Shuffler task to try to bring in the highest priority waiting task. The Shuffler does this by compacting memory and checkpointing a sufficient number of lower priority tasks that are not necessarily neighbors in the partition.

The checkpointing algorithm does have a limitation, however. If a large task is checkpointed and memory becomes fragmented by some smaller higher priority tasks, the smaller tasks block the large task from executing. The larger task can continue to be blocked until memory becomes free again by tasks exiting or being shuffled.

2.2.1.3 Checkpointing During Terminal Input Wait - Checkpointing during terminal input wait, a SYSGEN option, allows checkpointable tasks to be checkpointed while they are waiting for terminal input. This feature allows more copies of terminal I/O-bound tasks (for example, text editors) to run than normally could be run in a given amount of memory. This option frees memory for the long time periods while a user is thinking or between keystrokes. This is important for text editors that normally run at a high priority.

When the terminal driver dequeues a terminal input request for a checkpointable task that has checkpointing enabled, is not at AST state, and has ASTs enabled, the task is stopped from further execution. Thus, even if the task has not entered a wait state for the terminal input, its execution immediately stops when the request is dequeued by the terminal driver.

When a task is stopped in this manner, its effective priority within its partition drops to zero. (Its actual priority never changes.) Therefore, lower priority tasks ready to run can cause a higher priority task that is waiting for terminal input to be checkpointed. When the terminal input to the checkpointed higher-priority task is completed, the Executive removes the stop condition and the task can be brought back into memory. If necessary, the Executive displaces lower priority tasks to make room for it.

Normally, a task that was checkpointed for terminal input is not brought back into memory until its terminal request is satisfied. This is true even if memory becomes available during the wait. The only way the task can execute further, prior to the completion of the terminal input, is to receive an AST.

2.2.2 Disk Swapping

Disk swapping allows more tasks of equal priority to alternate the use of memory into which they cannot be loaded simultaneously. Swapping is accomplished by varying task priorities so that tasks of the same priority checkpoint each other periodically. Checkpointing is the only Executive feature required for swapping to operate.

Swapping does not affect the basic checkpointing algorithm as described under Checkpointing. For example, a task can only checkpoint another task of lower priority, never one of equal or higher priority. However, when swapping is enabled, the priority of tasks resident in memory varies with time (the installed priority of tasks remains unchanged - the swapping priority is for swapping only). The task's priority with respect to all of the other system resources does not change.

Two SYSGEN parameters control the swapping algorithm:

- Swapping interval. This parameter (S$$WPC) determines how often the Executive scans the partition lists to modify the swapping priority of resident tasks. A typical swapping interval might be about one-half second and is entered during SYSGEN as 30 (for 30 ticks or one-half second).

- Swapping priority range. This parameter (S$$WPR) is the absolute value of the range through which a task's priority varies from its installed priority. A typical value is 5. This value would cause a task's memory priority to vary from P+5 to P-5, where P is the priority set for the task when it was installed. The installed priority is in the word, L$BPRI, in the task image label block. The swapping priority is in the byte, H.SPRI, in the task header.

The key element of the swapping algorithm is the H.SPRI byte in the task's header, which is that task's swapping priority. The symbol, S$$WPR, is equated to the swapping priority range that is specified during SYSGEN. In a swapping system, each time a task is read into memory as the result of an initial task load or checkpoint read, the swapping priority byte in the task header is initialized to +S$$WPR (yielding a memory priority of the running priority plus the swapping priority). On the occurrence of each swapping interval, the swapping priority of each resident task is reduced by one until it reaches -S$$WPR (yielding a memory priority of the running priority minus the swapping priority). The Executive determines whether a nonresident task should checkpoint a resident task by comparing the running priority of the nonresident task with the sum of the running and swapping priorities of the resident task. If a possibility exists that checkpointing within a main partition might occur based on the new priorities, the Executive executes its partition allocation algorithm for that main partition.

The following points should be considered when specifying swapping parameters:

- The swapping interval should be approximately five times the round-robin scheduling interval. Round-robin scheduling is a SYSGEN option that periodically rotates the execution of tasks of equal priority that are in the System Task Directory (STD).

- From the time a task is loaded into memory, the average time (in clock ticks - 1 tick=1/60 of a second for a line frequency clock (1/50 of a second for 50 cycle machines) it takes for a task of the same running priority to checkpoint it is roughly equal to the product of the two swapping parameters.

- Tasks of the same running priority tend to get the same amount of time in memory. Tasks whose running priorities differ by less than the swapping range tend to receive different amounts of time in memory with the higher priority tasks getting much more time. When many tasks compete for memory and they are of different priorities, excessive checkpointing can occur. Therefore, tasks that use the swapping algorithm should have the same priority.

- In a system that supports checkpointing during terminal input, terminal input is also a factor in causing checkpointing to occur. Editors and other interactive tasks normally should run at a higher priority than more compute-bound tasks. The higher priority increases the terminal response time. Otherwise, the editor would have to compete with other tasks (for example, utilities) most of which run at a priority of 50. However, when an editor is waiting for terminal input, any lower priority task can checkpoint it. As soon as its input is complete, the editor can come back into memory by checkpointing the lower priority task. It is possible in a highly interactive system for the naturally high checkpoint rate to eliminate the need for the Executive swapping code to service many tasks of equal priority.

2.2.3 Shuffler (Memory Compaction)

$NXTSK, which is an Executive routine, attempts to find space in a partition for each waiting task. $NXTSK requests the Shuffler task after an allocation failure within the system-controlled partition takes place. The Shuffler receives no information from the Executive when it begins execution; it merely begins examining the partitions in the system starting from the beginning each time it is run. When the Shuffler encounters a system-controlled partition, the Shuffler makes two passes through the PCB list of subpartitions in an attempt to make room for the task or tasks in the main partition wait queue. In its first pass, the Shuffler attempts to remove all the holes in the partition. In addition, tasks that are stopped for terminal input are unconditionally checkpointed, if checkpointing during terminal input is included (a SYSGEN option).

Tasks that have been fixed in memory can be shuffled. However, the following occupants of a system-controlled partition cannot be shuffled. For this reason, these memory residents cause free space to be fragmented in the partition.

- Loaded device drivers

- Tasks that are connected to the ICS/ICR-11 or UDC-11 drivers

- A task whose partition was previously marked by the Shuffler as having a long outstanding I/O and whose I/O count has yet to drop to zero. Typical examples of this case are:

  - Tasks that issued terminal reads and cannot be checkpointed
  - LPA11 or tasks that have synchronous functions being serviced
  - Tasks using the Connect to Interrupt Vector directive

- Tasks that have been aborted and fixed by the Executive because of a memory parity error

- Dynamically-created common regions

When such a partition is encountered, the Shuffler treats the start of the subpartition as the end of an area within the main system-controlled partition. It treats each area as though it were a separate main partition. The tasks that occupy an area are shuffled until only one hole remains in the area.

In its second pass, the Shuffler creates a list of all the tasks that occupy an area of the main partition. This list is in reverse priority order. The Shuffler then examines this list in an attempt to find tasks that the waiting task can checkpoint. The Executive routine ($TSTCP) determines if the waiting task can checkpoint the task that owns the partition, which is the "owner task". If this scan of the list indicates that the sum of space available from holes and checkpointable tasks is sufficient for the waiting task, the Shuffler scans the list a second time to checkpoint (using $ICHKP) as many owner tasks as necessary.

To shuffle a task, the Shuffler first "freezes" the task in memory to prevent it from being checkpointed. Then, using the Executive routine, $BLXIO, it moves the task image in the partition. The Shuffler executes this move while in system state (thus preventing context switching) and it does the move in 256.-word blocks. Therefore, QIO speed optimizations with a large BLXIO transfer vector, a SYSGEN option, increases the response time for a very high priority real-time task.

The Shuffler's algorithm consists of two passes through the system-controlled partition; it executes the steps in each pass iteratively until the partition reaches a stable state.

2.2.3.1 **The Shuffler's First Pass** - In the first pass, the Shuffler starts at the beginning of the system-controlled partition and tries to move (shuffle) all tasks that are positioned above a gap of free space down to the base of the free space. When possible, it also checkpoints any tasks it encounters that are waiting for terminal input. Task shuffling occurs in the following steps:

- The Shuffler blocks the task from further execution and allocates the free space below the task.

- If necessary, the Shuffler waits for the task's outstanding I/O count to reach zero by checking it at intervals of approximately one-eighth of a second. If the task I/O count does not drop to zero in about one-half of a second, the Shuffler marks the task's partition as having long-outstanding I/O, deallocates the free space below the task, and restarts its first pass scan of the partition. The Executive clears the task's partition long-outstanding I/O indicator when it reduces the task's I/O count to zero.

- If the task's I/O count drops to zero within one-half of a second, the Shuffler moves the task down to the base of the free space. The speed of this move increases if the QIO speed optimizations were included during phase 1 of SYSGEN and a large BLXIO transfer vector was allocated.

- After the Shuffler completes the move, it unblocks the task to allow further execution and then it deallocates free space (now above the task).

When the Shuffler completes its first pass, all free space in the partition has been merged into one hole at the top of the partition. However, there may be additional holes below those tasks that cannot be shuffled.

2.2.3.2 **The Shuffler's Second Pass** - If the Shuffler completes its first pass and some tasks are still actively competing for memory in the partition, the Shuffler executes its second pass algorithm. In the second pass, the Shuffler creates an ascending, priority-ordered list of the tasks in the partition. It then uses this list to determine if the size of the waiting task is less than the sum of the free space in the partition and the size of one or more of the lower priority tasks that can be checkpointed. If the waiting task's size is less than this sum, the lower priority tasks are checkpointed. The Shuffler then restarts its first-pass algorithm to accumulate the freed-up space and allocate it to the waiting task. If tasks that cannot be shuffled fragment the system-controlled partition, the Shuffler executes the second-pass algorithm once for each fragment of the partition.

The RSX-11M philosophy of checkpointing in system-controlled partitions avoids preempting memory unless it can actually be used. When the Shuffler is active but not actually executing - for example, waiting for task I/O or a checkpointing operation to complete - it places itself in a state in which it may be checkpointed by any task. No lower priority task is checkpointed unless it is known beforehand that enough continuous space can be made available to load the higher priority task. The one exception is a checkpointable task waiting for terminal input. This task is swapped out unconditionally to make room for other tasks whenever the Shuffler is activated. It is not brought back in until the terminal input request is completed.

If the Shuffler completes its second pass without finding space for the waiting task, it searches for the next system-controlled partition in the system and exits if none exists.

2.2.4 The Loader (the System Loader Task)

The Loader, which is a resident RSX-11M system task, has three functions:

1. Reading a task, which is either about to start executing or is being fixed, into memory

2. Performing a checkpoint write of a task image from memory to disk

3. Performing a checkpoint read of a task image from disk into memory (resuming checkpointed tasks)

The Loader has the single objective of emptying its receive queue of tasks waiting for its attention.

The Loader's receive queue, which consists of a list of TCBs, is ordered by priority. When $NXTSK or one of its associated routines determines that action by the Loader is required, the TCB of the task to be moved to or from the disk is placed in the Loader's receive queue and the Executive requests Loader execution. After it has begun executing, the Loader examines two bits in the task status word of each TCB: the swap bit and the out-of-memory bit. These bits determine the Loader's action. The interpretation of these bits is as follows:

Swap Bit (TS.CKP)	Out-of-Memory Bit (TS.OUT)	Action
1	1	The task is read back into memory from its checkpoint area.
1	0	The task is written from memory into its checkpoint area.
0	1	The task is read into memory from its load image.
0	0	Illegal combination

When the Loader removes the next entry from its queue, it assumes memory is available if the task is about to be read (it has been allocated by $NXTSK). After the Loader writes a task into its swap area, it calls the release partition routine that in turn calls $NXTSK to select the next task that will occupy the partition.

The Loader performs several other functions associated with moving tasks to and from disk. These tasks include:

Task Load and Checkpoint Read

- Copy the task header into the Dynamic Storage Region
- Initialize task swapping priority
- Declare receive and receive-by-reference ASTs for the task if the appropriate queues are not empty

Checkpoint Read Only

- Release space held by the task within a dynamic checkpoint space file

Task Load Only

- Map the task's address windows to the task image and attach static commons

Checkpoint Write Only

- Deallocate the Dynamic Storage Region copy of the task's header
- Place the task's TCB in the partition wait queue
- Release the partition previously owned by the task. This results in a call to $NXTSK to reallocate the partition.

2.2.5 The $NXTSK Routine

$NXTSK is a routine in the Executive module, REQSB. $NXTSK works with both user- and system-controlled partitions and takes a single input: the PCB address of the partition to be reallocated. $NXTSK examines all the TCBs in the partition wait queue of the specified PCB and attempts to find space in the partition for each waiting task. When all TCBs in the queue have been examined, a return to the caller is executed.

$NXTSK is called by routines that need partitions; it assigns a partition to the highest priority task waiting to occupy the partition.

The inputs and functions of $NXTSK are listed in the following text.

2.2.5.1 **$NXTSK Inputs** - The only input to $NXTSK is the address, which is in register 0, of the PCB of the partition to assign.

2.2.5.2 **$NXTSK Functions** - $NXTSK has five possible functions:

1. The partition is not currently busy and a task is waiting to occupy the partition. $NXTSK assigns the partition to the waiting task and places a request on the Loader queue to load the task.

2. The partition is currently occupied by a task that is either of higher priority than all the waiting tasks or is not checkpointable. In this situation, $NXTSK cannot assign the partition to another task.

3. The partition is currently occupied by a lower priority checkpointable task. $NXTSK places a request in the Loader queue to checkpoint the task that owns the partition.

4. The highest priority task waiting to occupy the partition needs the main partition that is currently occupied by one or more tasks that are either of higher priority or are not checkpointable. In this situation, $NXTSK cannot assign the partition to another task.

5. The highest priority task waiting to occupy the partition requires the main partition that is currently occupied by one or more checkpointable tasks of lower priority. $NXTSK places a request in the Loader queue to checkpoint each task.

2.2.5.3 **$NXTSK Operation in a User-controlled Partition** - If the partition being reallocated is a task partition, $NXTSK first determines if the requested space in the partition is unused. If it is, the task is assigned the space and a request to read the task into memory is issued to the Loader.

If the space is being used by another task, $NXTSK calls $TSTCP to determine if the task or tasks occupying the partition can be checkpointed by the requesting task. If the task or tasks can be checkpointed by the requesting task and if checkpointing will produce enough space for the requesting task, $NXTSK calls $ICHKP to initiate a checkpoint of the task or tasks that occupy the space.

When the Loader completes the checkpoint write of the tasks that occupy the partition, it calls $NXTSK again to allow the requesting task to find the newly vacated space in the partition.

2.2.5.4 $NXTSK Operation in a System-controlled Partition - If the partition being reallocated is a system-controlled partition, $NXTSK calls $FNDSP to find a hole in the partition large enough for the requesting task. If such a hole is found, $NXTSK assigns to the requesting task as much of the space in the hole as it requires, and issues a request to the Loader to read the task into memory.

If $NXTSK cannot find a satisfactory hole, it searches for a contiguous combination of holes and checkpointable tasks large enough to form a single hole large enough for the requesting task. $NXTSK calls $TSTCP to determine if the memory resident task or tasks can be checkpointed by the requesting task. If such a combination is found, $NXTSK calls $ICHKP to initiate a checkpoint of each task in the potential hole. $NXTSK then returns to the caller, relying on the Loader to call it again when checkpointing is complete.

Finally, if no combination of contiguous holes and checkpointable tasks can be found, and if the Shuffler is installed, $NXTSK requests execution of the Shuffler.

2.2.6 Routines That Call $NXTSK

Examining the routines that call $NXTSK yields information on the circumstances under which reallocation of a partition occurs. Figure 2-2 shows that such routines tend to fall into two groups. Routines in the first group call $NXTSK directly (1 through 5 and A through G in Figure 2-2). Routines in the second group call routines A through G. The numbers and letters in Figure 2-2 are an index into a table of short routine descriptions that follow the Figure.

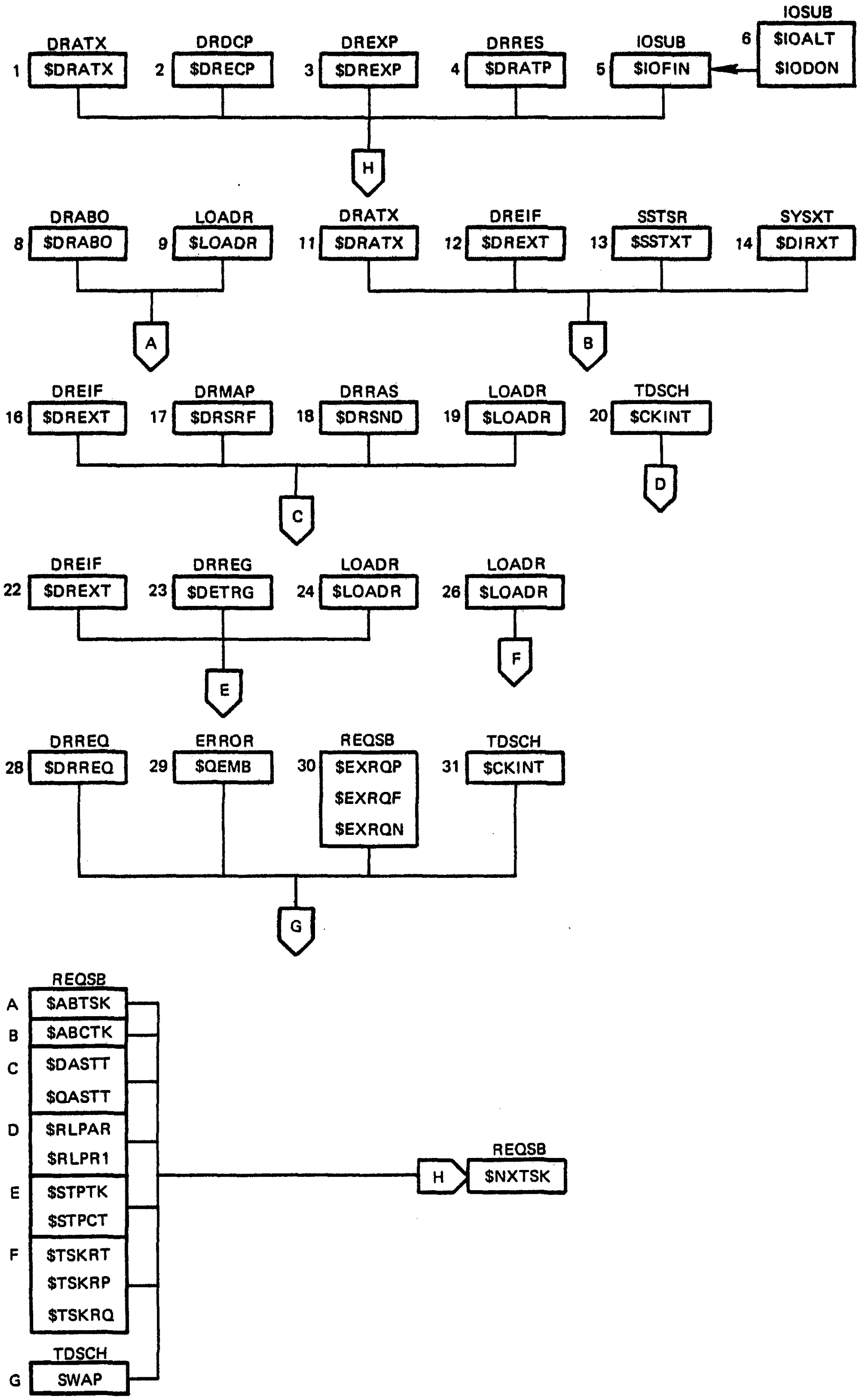


Figure 2-2 Routines That Call $NXTSK

2.2.6.1 **Routine Description** - The numbers and letters in the following text are keyed to the numbers and letters in Figure 2-2.

1. $DRATX - DRATX Module - AST exit directive
   Calls $NXTSK if task is stopped at task state.

2. $DRECP - DRECP Module - Enable Checkpointing directive
   Calls $NXTSK when checkpointable task enables checkpointing

3. $DREXP - DREXP Module - Extend Task directive
   Returns to $NXTSK to reallocate the partition after changing the P.SIZE word in the task's PCB.

4. $DRATP - DRRES Module - Alter Task Priority directive
   Calls $NXTSK to reallocate the partition due to changed task priority.

5. $IOFIN - IOSUB Module - I/O finish
   Calls $NXTSK after completion of what has previously been declared a long-outstanding I/O. An I/O operation is declared a long I/O by the Shuffler if the operation does not finish in 1/2 second.

   Routine that calls $IOFIN:

   6. $IODON/$IOALT - IOSUB Module - I/O done
      Calls $IOFIN unconditionally. Most drivers call $IOALT, $IODON, or $IOFIN.

A. $ABTSK - REQSB Module - Abort task by TCB address
   Calls $NXTSK after the task has been marked for abort.

   Routines that call $ABTSK:

   8. $DRABO - DRABO Module - Abort Task directive
      Calls $ABTSK to actually abort the task.

   9. $LOADR - LOADR Module - Task Loader
      Calls $ABTSK if a disk read error occurred while reading task.

B. $ABCTK - REQSB Module - Abort current task
   Calls $ABTSK to abort task.

   Routines that call $ABCTK:

   11. $DRATX - DRATX Module - AST Exit directive
       Calls $ABCTK when address check of task stack fails; $ABCTK aborts the task and the next AST is not acted upon.

   12. $DREXT - DREIF Module - Exit directive
       Calls $ABCTK if task is already marked for abort.

   13. SSTXT - SSTSR Module - Common SST processing routine
       Calls $ABCTK if task has no SST vector.

   14. $DIRXT - SYSXT Module - Directive exit (context switch)
       Call $ABCTK if address check of task fails.

C. $DASTT/$QASTT - REQSB Module - Declare or Queue (non-I/O) AST to task.
   $QASTT calls $NXTSK if task is not stopped; $DASTT calls $QASTT.

Routines that call $DASTT:

16. $DREXT - DREIF Module - Exit directive
    Calls $DASTT to act upon floating-point, powerfail, and receive ASTs on task exit.

17. $DRSRF - DRMAP Module - Send-by-Reference directive
    Calls $DASTT to act upon receive-by-reference AST for receiver.

18. $DRSND - DRRAS Module - Send Data directive
    Calls $DASTT to act upon receive AST for receiver.

19. $LOADR - LOADR Module - Task loader
    Calls $DASTT for receive and receive-by-reference ASTs if task queues are not empty when task is brought into memory.

Routine that calls $QASTT:

20. $CKINT - TDSCH Module - Clock interrupt service routine
    Calls $QASTT when mark time request has expired.

D. $RLPAR/$RLPR1 - REQSB Module - Release partition
   Calls $NXTSK to allow next highest priority task to occupy partition.

Routines that call $RLPAR/$RLPR1:

22. DREXT - DREIF Module - Exit directive
    Calls $RLPAR when task exit is complete.

23. $DETRG - DRREG Module - Detach Region directive
    Calls $RLPR1 when a dynamic common region with a "delete on last detach" attribute is detached by the last task and is deallocated from a system-controlled partition.

24. $LOADR - LOADR Module - Task loader
    Calls $RLPAR upon completion of a checkpoint write of a task.

E. $STPCT/$STPTK - REQSB Module - Stop (current) task
   Calls $NXTSK after setting task's stop bit.

Routine that calls $STPCT/$STPTK:

26. $LOADR - LOADR Module - Task loader
    Calls $STPCT to set its own stop bit when waiting for work.
    Note: Many privileged system tasks call $STPCT.

F. $TSKRT/$TSKRP/$TSKRQ - REGSB Module - Request task execution
   Calls $NXTSK to force reallocation of task's partition, if task is not fixed or active.

Routines that call $TSKRT/$TSKRP/$TSKRQ

28. $DRREQ - DRREQ Module - Task Request directive
    Calls $TSKRP to request desired task.

29. $QEMB - ERROR Module - Queue error message block
    Calls $TSKRT to request error logger to run.

30. $EXRQP/$EXRQF/$EXRQN - REQSB Module - Execute task request
    Calls $TSKRT after clearing task's stop bit on internal executive request of task (TKTN, LOADR, etc).

31. $CKINT - TDSCH Module - Clock interrupt service routines
    Calls $TSKRT to carry out a schedule request after a clock queue has expired.

G. SWAP - TDSCH Module - Disk swapping algorithm
Calls $NXTSK during scan of all partitions in system. $NXTSK is called:

- Whenever a system-controlled partition is encountered

- Whenever the swapping priority of a task is reduced

2.2.7 Routines That $NXTSK Calls

2.2.7.1 $CHKPT Routine - $CHKPT checkpoints a task. When an Executive routine calls $CHKPT, all the conditions necessary for a task to be checkpointable have been met. If dynamic allocation of checkpoint space has been selected as a SYSGEN option, $CHKPT searches the checkpoint files for available space by using the $FNDSP routine. If this search fails, or if dynamic allocation of checkpoint space was not a selected option, $CHKPT determines if checkpoint space is allocated within the task's disk image file. If checkpoint space is found in either the dynamic checkpoint space or the task image file, $CHKPT issues a checkpoint request to the Loader and requests execution of the Loader.

If no checkpoint space can be found, $CHKPT requests the TKTN task to print a message that so informs the user.

The method used by the Executive to allocate space within the checkpoint file is identical to the method used to allocate space within a system-controlled partition. In both cases, a main PCB describes the overall area of allocation (main partition or entire disk file) and a sub-PCB describes a single fragment of the area (sub-partition for the task or portion of the checkpoint file reserved for one task image). In addition, both algorithms use $FNDSP to find a hole of the required size within the area described by the main PCB.

2.2.8 $FNDSP Routine

$FNDSP tries to find space in a PCB list in a system-controlled partition for a PCB. It searches through the list until it finds a large enough hole. If it finds a large enough hole, $FNDSP links the PCB into the list. If it does not find a hole, $FNDSP sets the C-bit before returning to the calling routine.

2.2.9 $ICHKP Routine

Other routines call $ICHKP to begin the checkpointing process for a task that owns a partition. $ICHKP has three possible functions:

1. If the task is already being checkpointed, $ICHKP immediately executes a return to the calling routine.

2. If the task is being read into memory or has outstanding I/O, $ICHKP marks the task for checkpointing. If the task is being read into memory, the Loader detects that the task is marked for checkpointing when the read is done and immediately checkpoints the task. If the task has I/O outstanding, $IOFIN in the IOSUB module is entered when the I/O is complete. $IOFIN detects the checkpoint request and checkpoints the task. However, terminal I/O that has been buffered is not included in the I/O count of a task.

3. If neither 1. nor 2. above are true, $ICHKP calls $CHKPT to checkpoint the task.

2.2.10 $TSTCP Routine

Other routines call $TSTCP to determine if a task that owns space in a partition (owner task) can be checkpointed by a task that is requesting space in the same partition (requesting task). There are two conditions that must be met before the requesting task can checkpoint the owner task.

First, the owner task must be eligible for checkpointing. A task is eligible for checkpointing if:

- It is neither fixed nor being fixed

- It was taskbuilt or installed as checkpointable

- It does not have checkpointing disabled

The second condition for checkpointability is that the requesting task must have a default priority higher than the effective priority of the owner task. Two factors may make the effective priority of a memory-resident task different from its default priority:

1. Tasks that are stopped, or that are stopped for terminal input and do not have outstanding ASTs, have an effective priority of zero.

2. In a system that includes the disk swapping of equal or nearly equal priority tasks, the effective priority of a task that is in memory is the sum of its default priority (from the TCB) and its swapping priority (from the task header).

If these two conditions are met, the calling routine is informed that the requesting task can checkpoint the owner task. Note that the presence of outstanding I/O belonging to the owner task is not detected by this routine.

2.3 MEMORY ALLOCATION FLOW DIAGRAMS

Memory allocation within partitions is managed by two types of routines in RSX-11M. Routines in the first group detect the presence of conditions that indicate reallocation of space within a partition is necessary.

Routines in the second group (headed by the $NXTSK routine) find or create space for a task within a partition, and load the task into that space. In other words, the first group of routines determines when memory allocation should take place and the second group of routines does the reallocation.

This flow diagram section contains flow diagrams of $NXTSK and the routines that it invokes. It also contains flow diagrams of the Loader, Shuffler, $ALOCB, and $DEACB.

2.3.1 $ALCLK Logical Flow Diagram

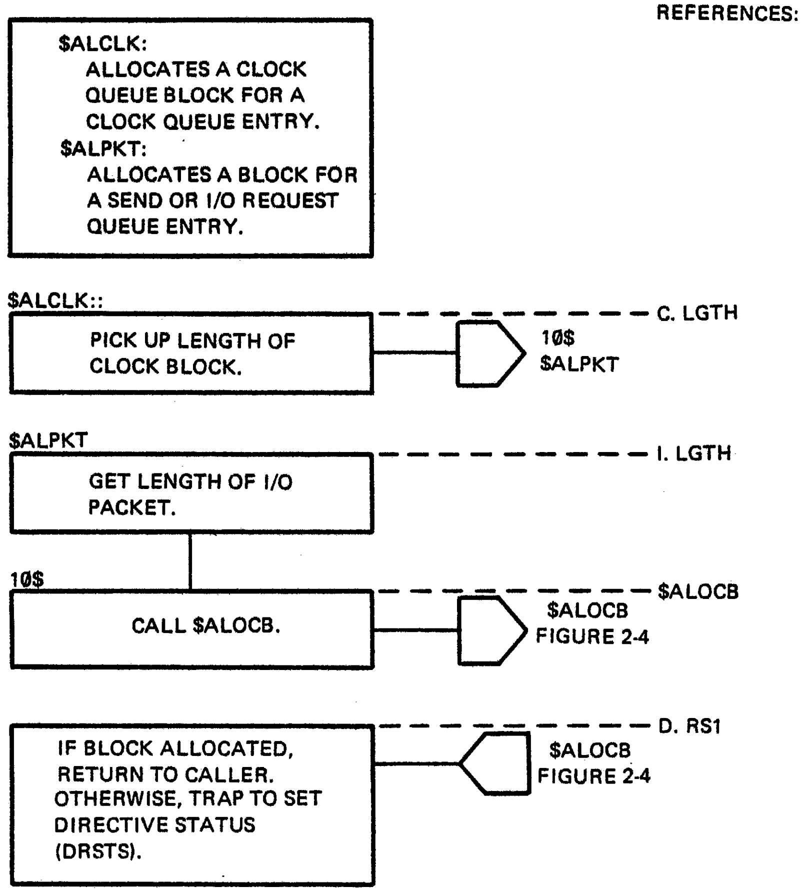

REFERENCES:
$ALCLK:
ALLOCATES A CLOCK QUEUE BLOCK FOR A CLOCK QUEUE ENTRY.
$ALPKT:
ALLOCATES A BLOCK FOR A SEND OR I/O REQUEST QUEUE ENTRY.
$ALCLK::
PICK UP LENGTH OF CLOCK BLOCK.
C. LGTH
10$
$ALPKT
$ALPKT
GET LENGTH OF I/O PACKET.
I. LGTH
10$
CALL $ALOCB.
$ALOCB
$ALOCB
FIGURE 2-4
IF BLOCK ALLOCATED, RETURN TO CALLER. OTHERWISE, TRAP TO SET DIRECTIVE STATUS (DRSTS).
D. RS1
$ALOCB
FIGURE 2-4

Figure 2-3 $ALCLK Logical Flow Diagram

2.3.2 $ALOCB Logical Flow Diagram

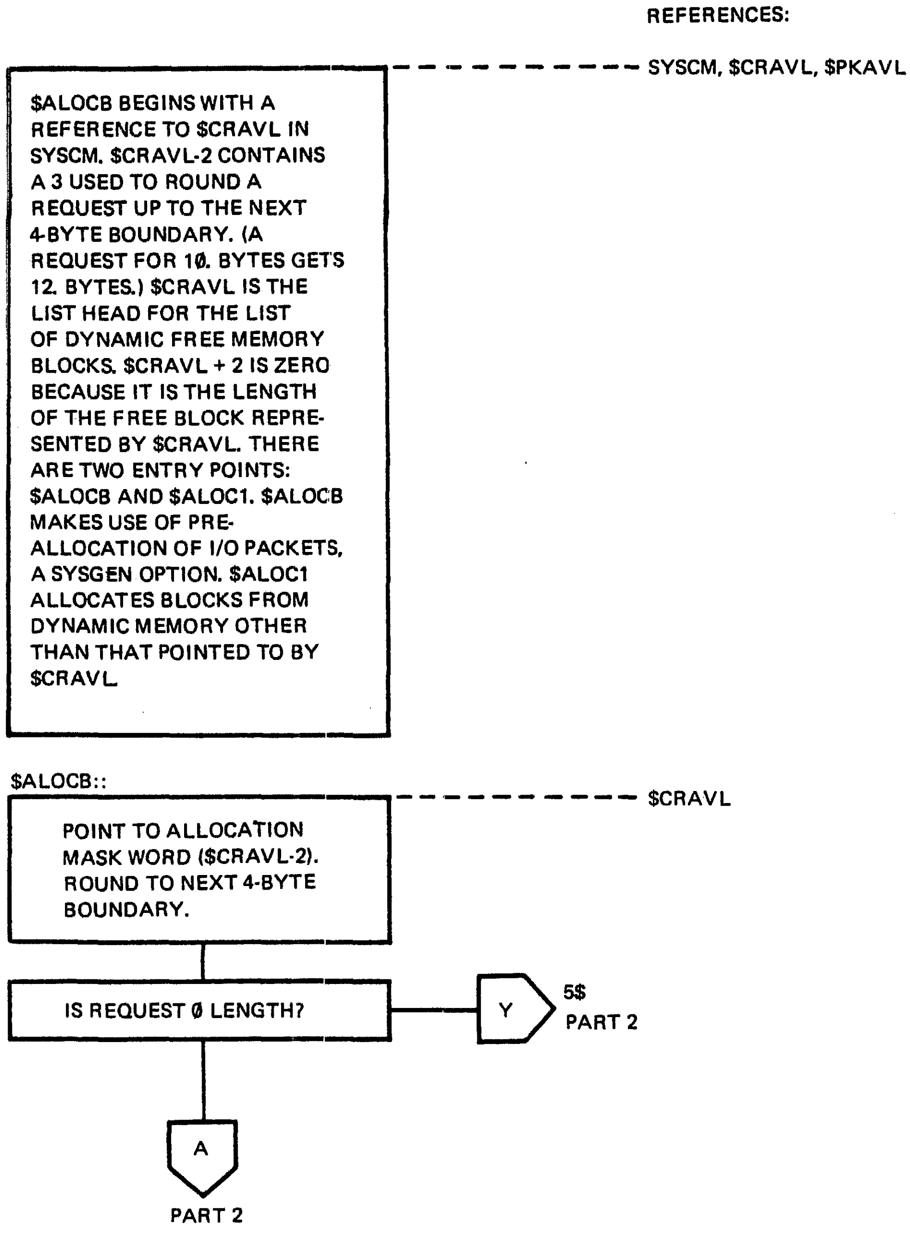

REFERENCES:
SYSCM, $CRAVL, $PKAVL
$ALOCB BEGINS WITH A REFERENCE TO $CRAVL IN SYSCM. $CRAVL-2 CONTAINS A 3 USED TO ROUND A REQUEST UP TO THE NEXT 4-BYTE BOUNDARY. (A REQUEST FOR 1Ø. BYTES GETS 12. BYTES.) $CRAVL IS THE LIST HEAD FOR THE LIST OF DYNAMIC FREE MEMORY BLOCKS. $CRAVL + 2 IS ZERO BECAUSE IT IS THE LENGTH OF THE FREE BLOCK REPRESENTED BY $CRAVL. THERE ARE TWO ENTRY POINTS: $ALOCB AND $ALOC1. $ALOCB MAKES USE OF PRE-ALLOCATION OF I/O PACKETS, A SYSGEN OPTION. $ALOC1 ALLOCATES BLOCKS FROM DYNAMIC MEMORY OTHER THAN THAT POINTED TO BY $CRAVL.
$ALOCB::
$CRAVL
POINT TO ALLOCATION MASK WORD ($CRAVL-2). ROUND TO NEXT 4-BYTE BOUNDARY.
IS REQUEST Ø LENGTH?
Y
5$
PART 2
A
PART 2

Figure 2-4 $ALOCB Logical Flow Diagram (Part 1 of 2)

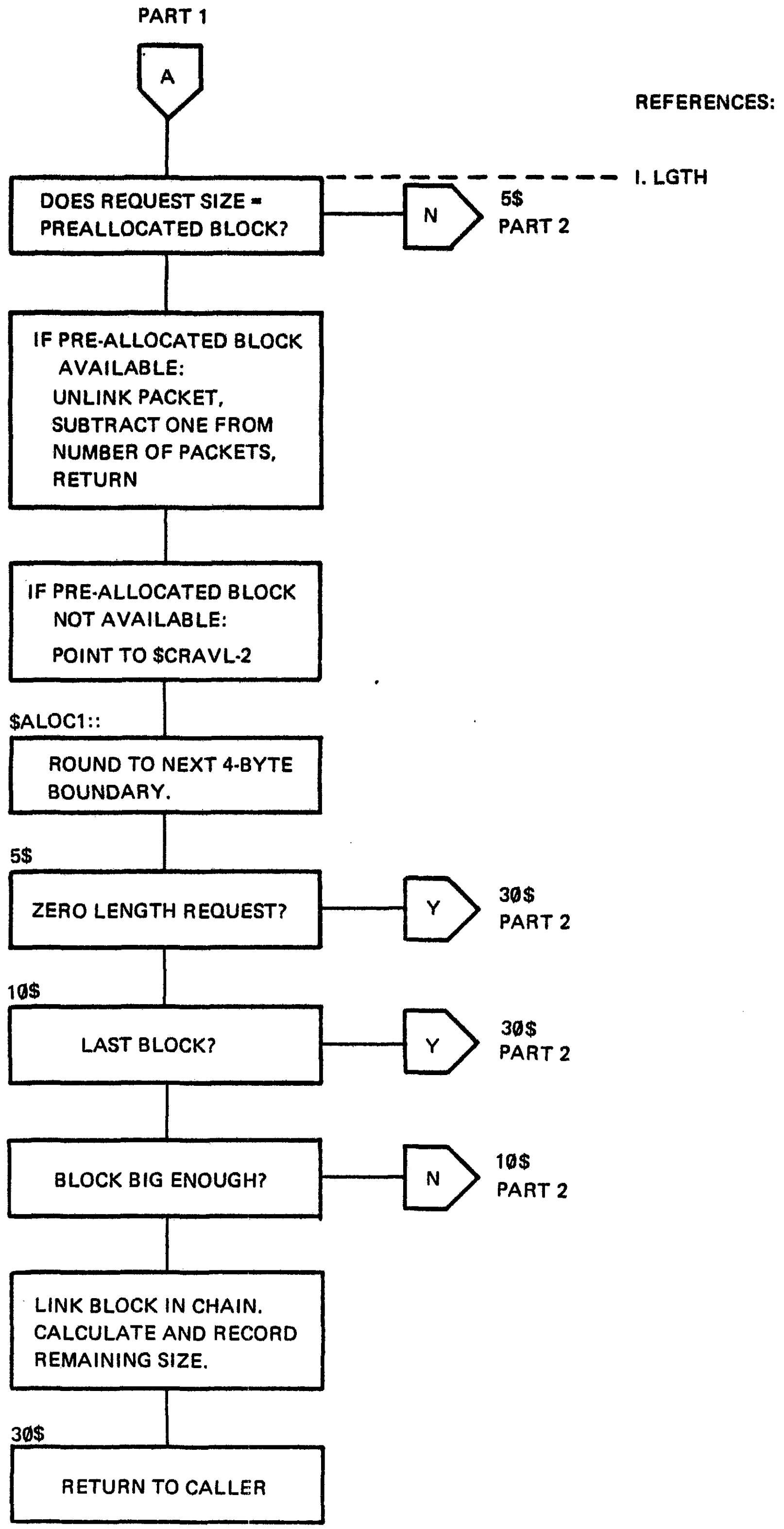

PART 1
A
REFERENCES:
I. LGTH
DOES REQUEST SIZE = PREALLOCATED BLOCK?
N
5$
PART 2
IF PRE-ALLOCATED BLOCK AVAILABLE:
UNLINK PACKET,
SUBTRACT ONE FROM NUMBER OF PACKETS,
RETURN
IF PRE-ALLOCATED BLOCK NOT AVAILABLE:
POINT TO $CRAVL-2
$ALOC1::
ROUND TO NEXT 4-BYTE BOUNDARY.
5$
ZERO LENGTH REQUEST?
Y
30$
PART 2
10$
LAST BLOCK?
Y
30$
PART 2
BLOCK BIG ENOUGH?
N
10$
PART 2
LINK BLOCK IN CHAIN.
CALCULATE AND RECORD REMAINING SIZE.
30$
RETURN TO CALLER

Figure 2-4 $ALOCB Logical Flow Diagram (Part 2 of 2)

2.3.3 $CHKPT Logical Flow Diagram

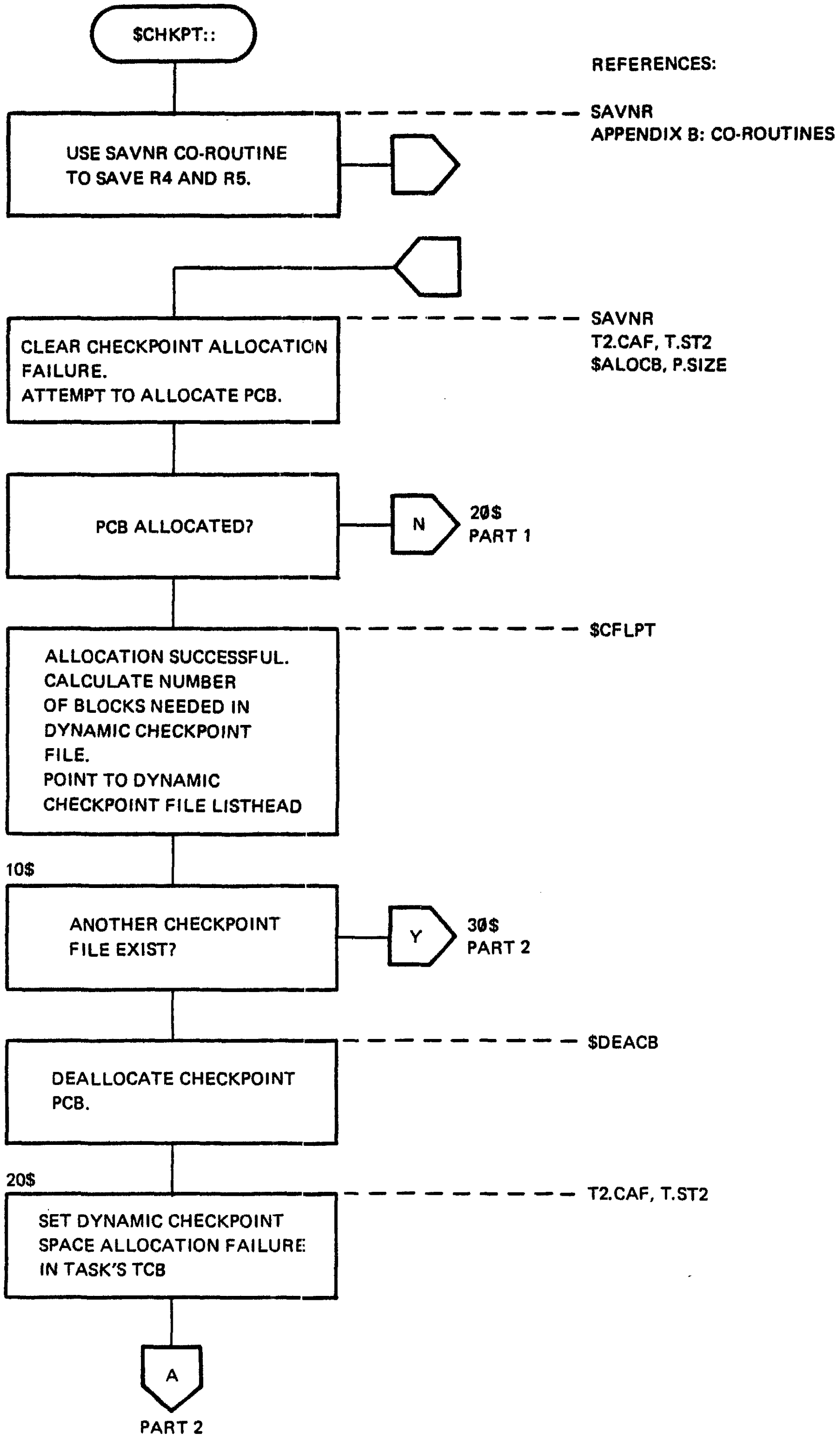

$CHKPT::
REFERENCES:
SAVNR
APPENDIX B: CO-ROUTINES
USE SAVNR CO-ROUTINE TO SAVE R4 AND R5.
SAVNR
T2.CAF, T.ST2
$ALOCB, P.SIZE
CLEAR CHECKPOINT ALLOCATION FAILURE.
ATTEMPT TO ALLOCATE PCB.
PCB ALLOCATED?
N
20$
PART 1
$CFLPT
ALLOCATION SUCCESSFUL.
CALCULATE NUMBER OF BLOCKS NEEDED IN DYNAMIC CHECKPOINT FILE.
POINT TO DYNAMIC CHECKPOINT FILE LISTHEAD
10$
ANOTHER CHECKPOINT FILE EXIST?
Y
30$
PART 2
$DEACB
DEALLOCATE CHECKPOINT PCB.
20$
T2.CAF, T.ST2
SET DYNAMIC CHECKPOINT SPACE ALLOCATION FAILURE IN TASK'S TCB
A
PART 2

Figure 2-5 $CHKPT Logical Flow Diagram (Part 1 of 2)

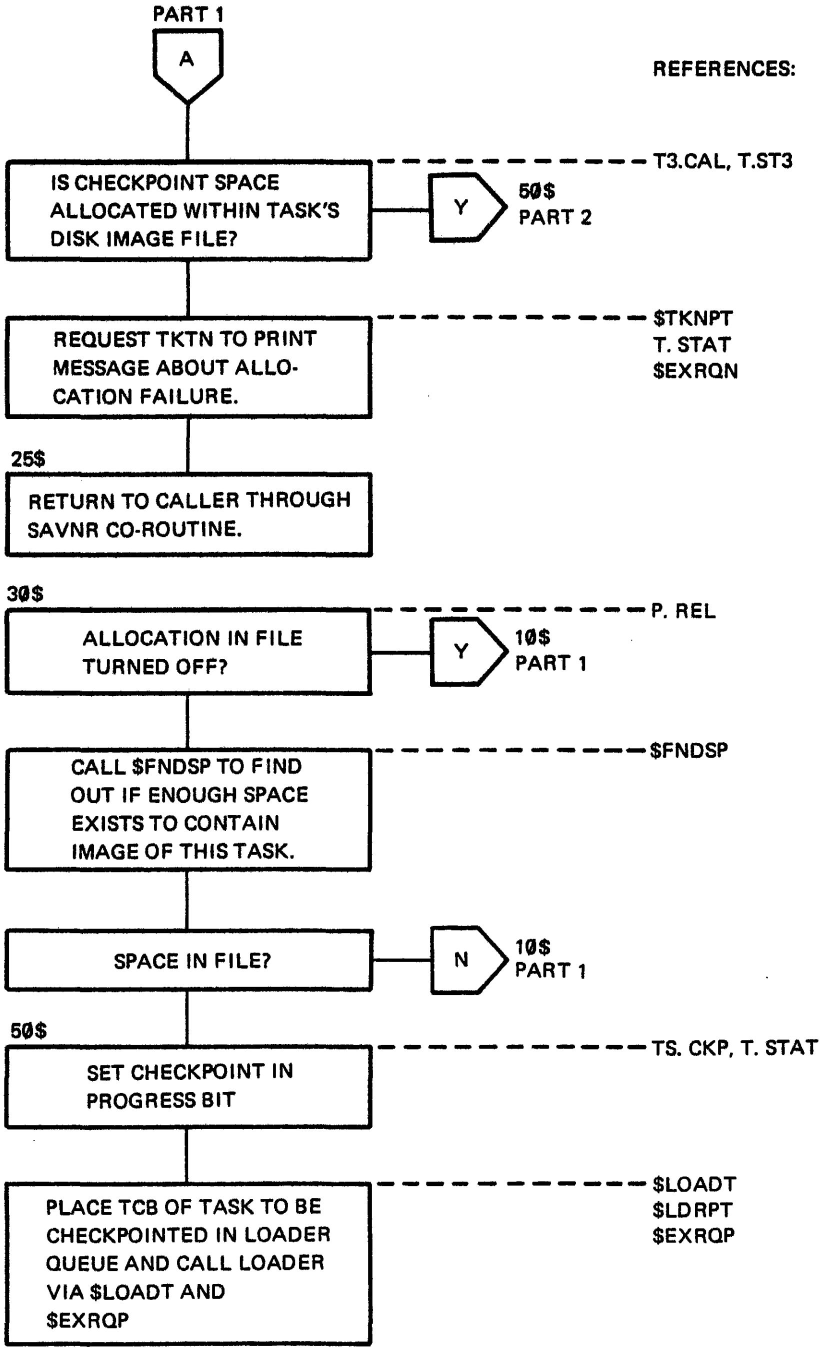
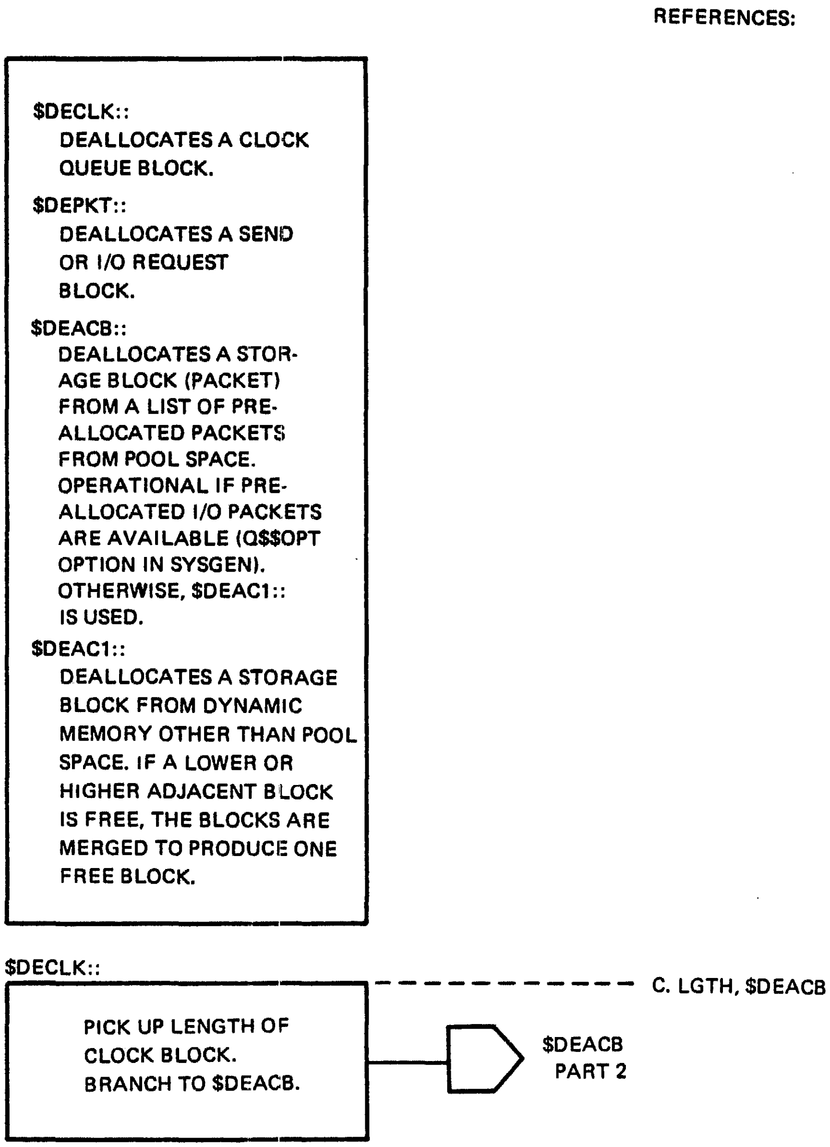

PART 1
A
REFERENCES:
IS CHECKPOINT SPACE ALLOCATED WITHIN TASK'S DISK IMAGE FILE?
T3.CAL, T.ST3
Y
5Ø$
PART 2
REQUEST TKTN TO PRINT MESSAGE ABOUT ALLO-CATION FAILURE.
$TKNPT
T. STAT
$EXRQN
25$
RETURN TO CALLER THROUGH SAVNR CO-ROUTINE.
3Ø$
ALLOCATION IN FILE TURNED OFF?
P. REL
Y
1Ø$
PART 1
CALL $FNDSP TO FIND OUT IF ENOUGH SPACE EXISTS TO CONTAIN IMAGE OF THIS TASK.
$FNDSP
SPACE IN FILE?
N
1Ø$
PART 1
5Ø$
SET CHECKPOINT IN PROGRESS BIT
TS. CKP, T. STAT
PLACE TCB OF TASK TO BE CHECKPOINTED IN LOADER QUEUE AND CALL LOADER VIA $LOADT AND $EXRQP
$LOADT
$LDRPT
$EXRQP

Figure 2-5 $CHKPT Logical Flow Diagram (Part 2 of 2)

2.3.4 $DECLK-$DEPKT-$DEACB Logical Flow Diagram



$DECLK::
DEALLOCATES A CLOCK QUEUE BLOCK.

$DEPKT::
DEALLOCATES A SEND OR I/O REQUEST BLOCK.

$DEACB::
DEALLOCATES A STORAGE BLOCK (PACKET) FROM A LIST OF PRE-ALLOCATED PACKETS FROM POOL SPACE. OPERATIONAL IF PRE-ALLOCATED I/O PACKETS ARE AVAILABLE (Q$$OPT OPTION IN SYSGEN). OTHERWISE, $DEAC1:: IS USED.

$DEAC1::
DEALLOCATES A STORAGE BLOCK FROM DYNAMIC MEMORY OTHER THAN POOL SPACE. IF A LOWER OR HIGHER ADJACENT BLOCK IS FREE, THE BLOCKS ARE MERGED TO PRODUCE ONE FREE BLOCK.

Figure 2-6 $DECLK-$DEPKT-$DEACB Logical Flow Diagram (Part 1 of 5)

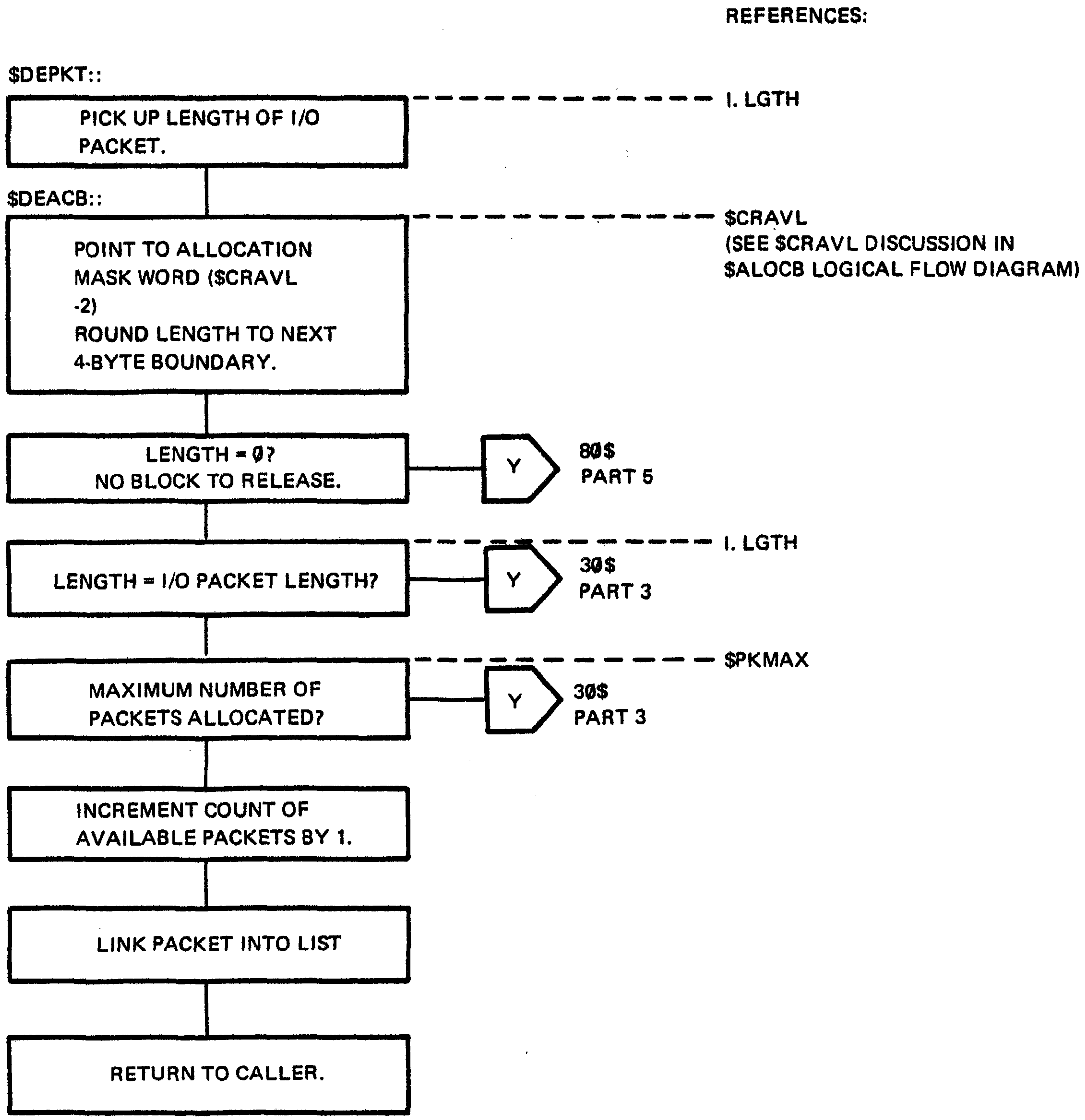

REFERENCES:
$DEPKT::
PICK UP LENGTH OF I/O PACKET.
I. LGTH
$DEACB::
POINT TO ALLOCATION MASK WORD ($CRAVL -2)
ROUND LENGTH TO NEXT 4-BYTE BOUNDARY.
$CRAVL
(SEE $CRAVL DISCUSSION IN $ALOCB LOGICAL FLOW DIAGRAM)
LENGTH = 0?
NO BLOCK TO RELEASE.
Y
80$
PART 5
LENGTH = I/O PACKET LENGTH?
I. LGTH
Y
30$
PART 3
MAXIMUM NUMBER OF PACKETS ALLOCATED?
$PKMAX
Y
30$
PART 3
INCREMENT COUNT OF AVAILABLE PACKETS BY 1.
LINK PACKET INTO LIST
RETURN TO CALLER.

Figure 2-6 $DECLK-$DEPKT-$DEACB Logical Flow Diagram (Part 2 of 5)

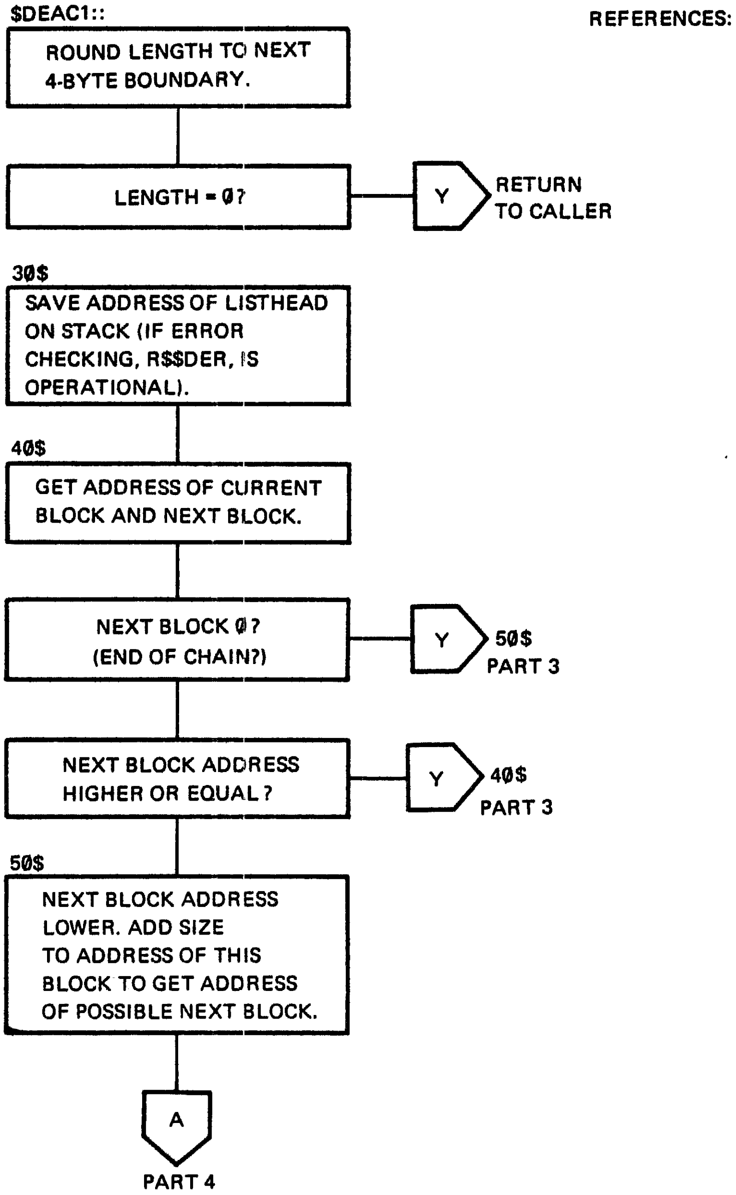

$DEAC1::
REFERENCES:
ROUND LENGTH TO NEXT 4-BYTE BOUNDARY.
LENGTH = Ø?
Y
RETURN TO CALLER
30$
SAVE ADDRESS OF LISTHEAD ON STACK (IF ERROR CHECKING, R$$DER, IS OPERATIONAL).
40$
GET ADDRESS OF CURRENT BLOCK AND NEXT BLOCK.
NEXT BLOCK Ø ? (END OF CHAIN?)
Y
50$ PART 3
NEXT BLOCK ADDRESS HIGHER OR EQUAL ?
Y
40$ PART 3
50$
NEXT BLOCK ADDRESS LOWER. ADD SIZE TO ADDRESS OF THIS BLOCK TO GET ADDRESS OF POSSIBLE NEXT BLOCK.
A
PART 4

Figure 2-6 $DECLK-$DEPKT-$DEACB Logical Flow Diagram (Part 3 of 5)

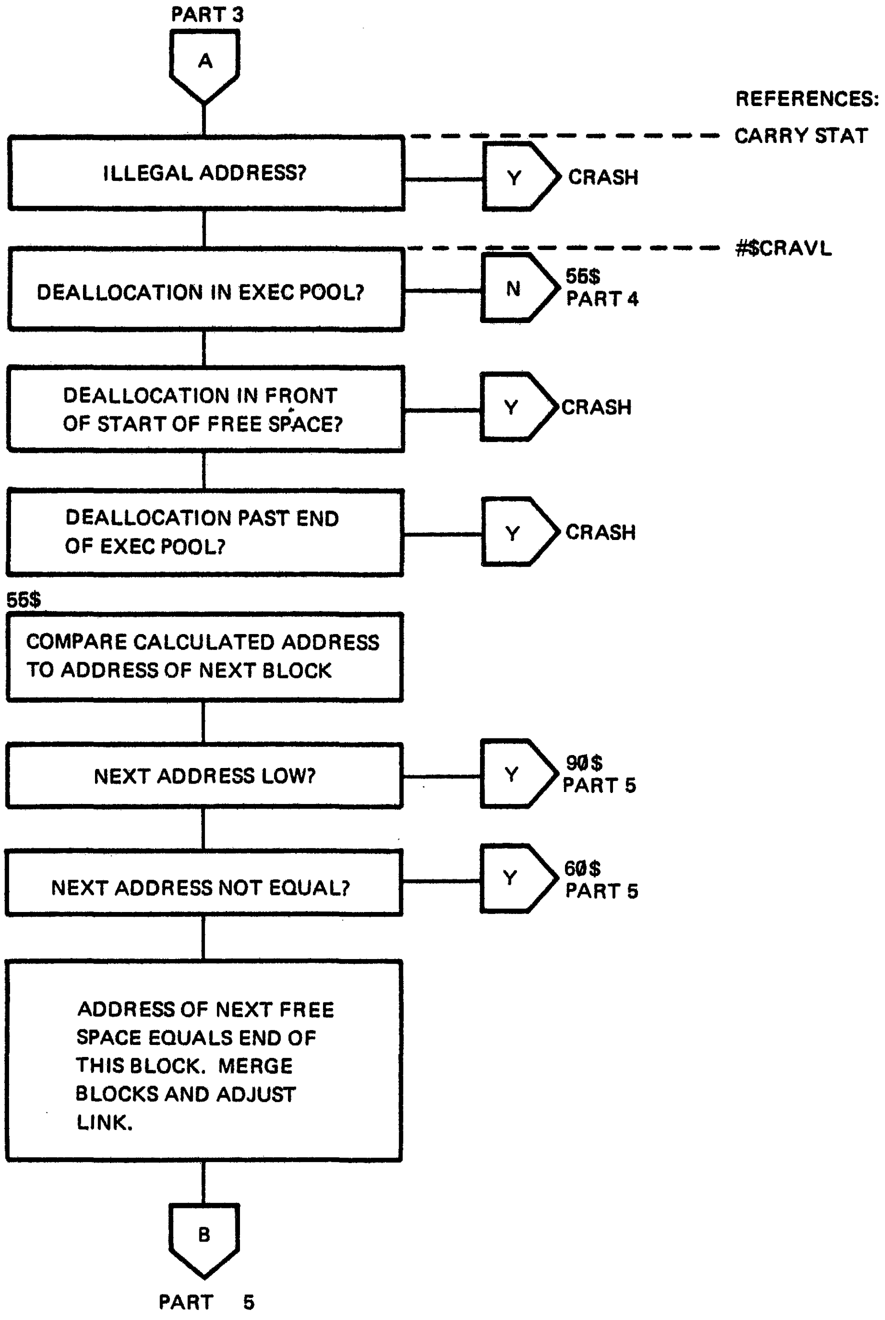

PART 3
A
REFERENCES:
CARRY STAT
ILLEGAL ADDRESS?
Y
CRASH
#$CRAVL
DEALLOCATION IN EXEC POOL?
N
55$
PART 4
DEALLOCATION IN FRONT OF START OF FREE SPACE?
Y
CRASH
DEALLOCATION PAST END OF EXEC POOL?
Y
CRASH
55$
COMPARE CALCULATED ADDRESS TO ADDRESS OF NEXT BLOCK
NEXT ADDRESS LOW?
Y
90$
PART 5
NEXT ADDRESS NOT EQUAL?
Y
60$
PART 5
ADDRESS OF NEXT FREE SPACE EQUALS END OF THIS BLOCK. MERGE BLOCKS AND ADJUST LINK.
B
PART 5

Figure 2-6 $DECLK-$DEPKT-$DEACB Logical Flow Diagram (Part 4 of 5)

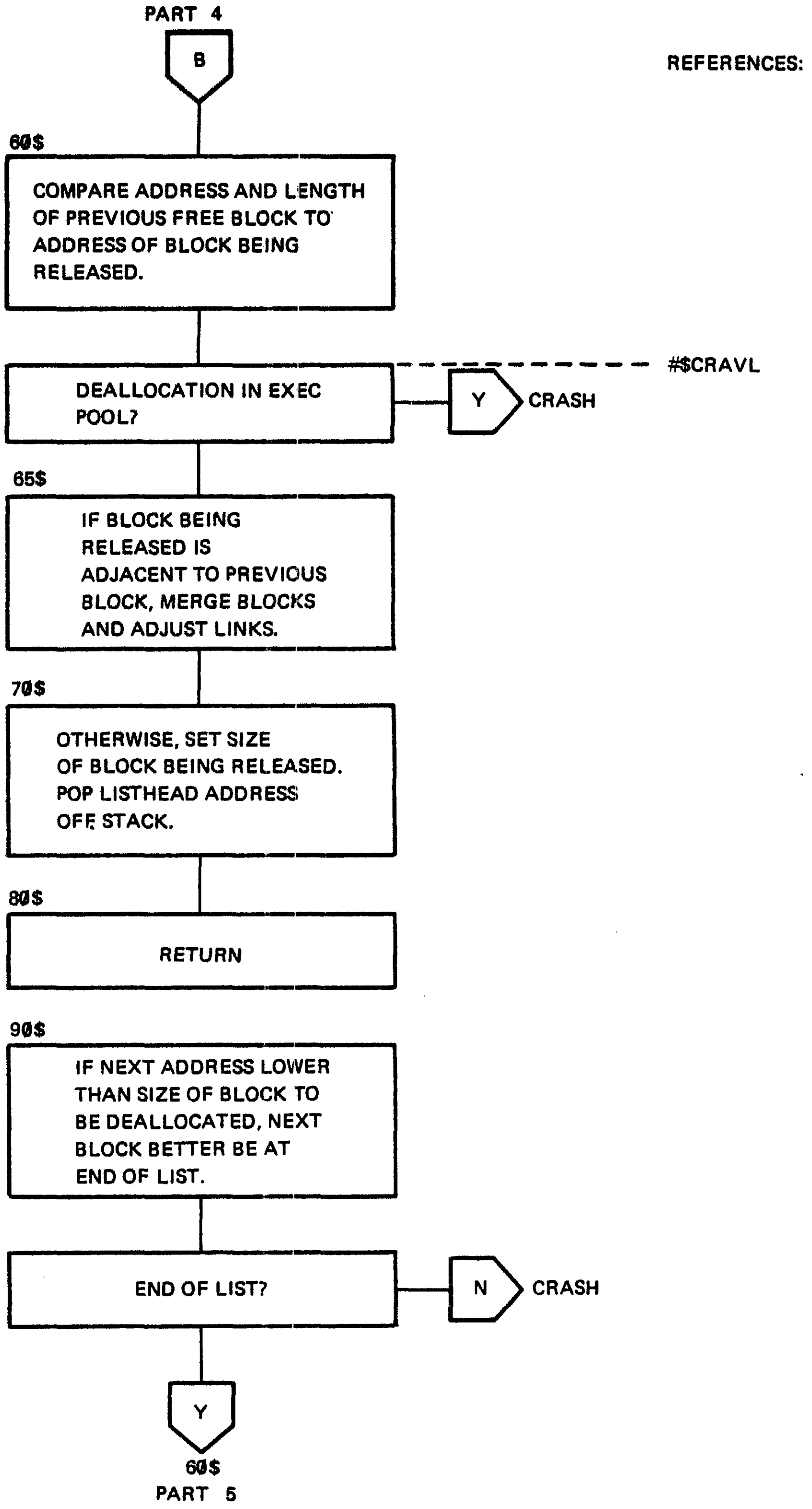

PART 4
B
REFERENCES:
60$
COMPARE ADDRESS AND LENGTH OF PREVIOUS FREE BLOCK TO ADDRESS OF BLOCK BEING RELEASED.
#$CRAVL
DEALLOCATION IN EXEC POOL?
Y
CRASH
65$
IF BLOCK BEING RELEASED IS ADJACENT TO PREVIOUS BLOCK, MERGE BLOCKS AND ADJUST LINKS.
70$
OTHERWISE, SET SIZE OF BLOCK BEING RELEASED. POP LISTHEAD ADDRESS OFF STACK.
80$
RETURN
90$
IF NEXT ADDRESS LOWER THAN SIZE OF BLOCK TO BE DEALLOCATED, NEXT BLOCK BETTER BE AT END OF LIST.
END OF LIST?
N
CRASH
Y
60$
PART 5

Figure 2-6 $DECLK-$DEPKT-$DEACB Logical Flow Diagram (Part 5 of 5)

2.3.5 $FNDSP Logical Flow Diagram

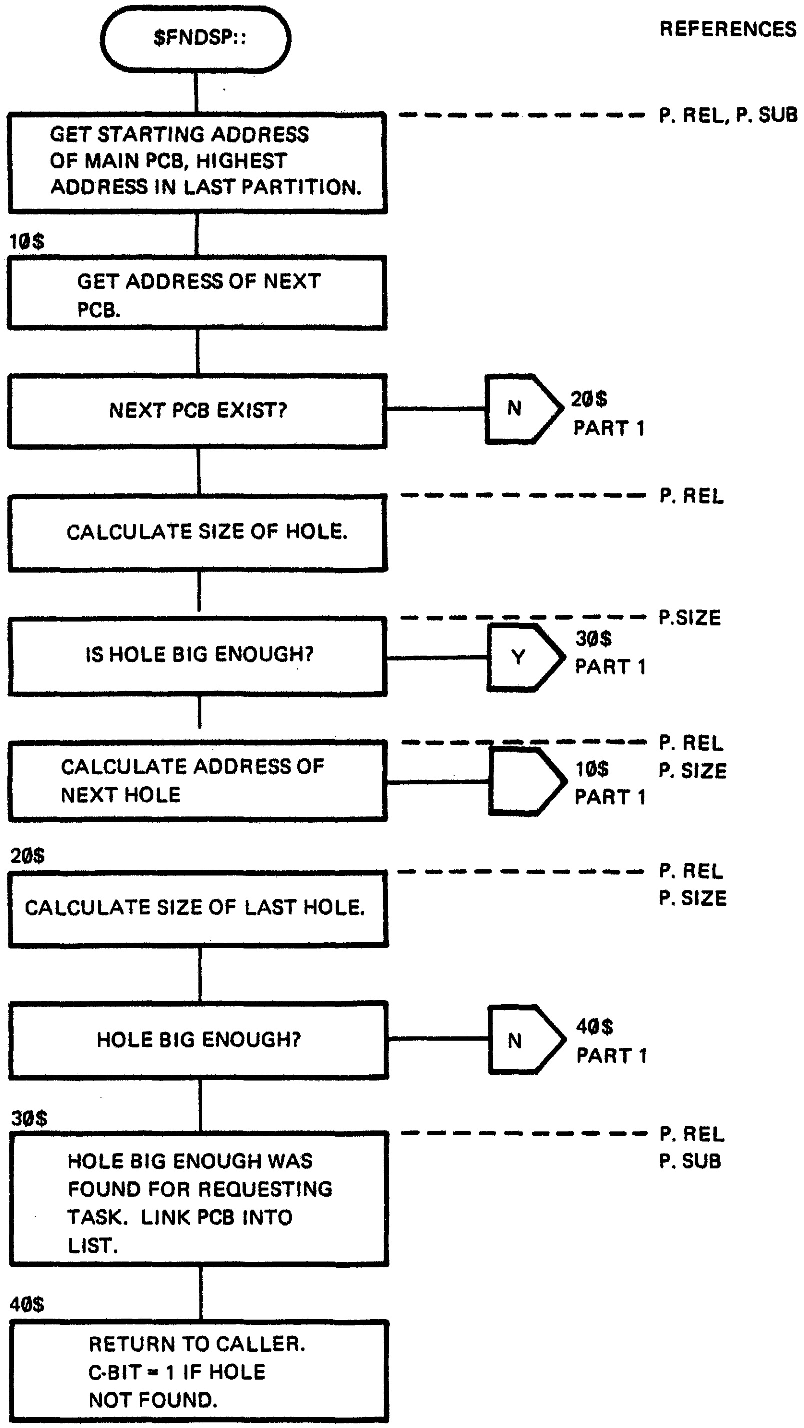

REFERENCES
$FNDSP::
GET STARTING ADDRESS OF MAIN PCB, HIGHEST ADDRESS IN LAST PARTITION.
P. REL, P. SUB
10$
GET ADDRESS OF NEXT PCB.
NEXT PCB EXIST?
N
20$ PART 1
P. REL
CALCULATE SIZE OF HOLE.
P.SIZE
IS HOLE BIG ENOUGH?
Y
30$ PART 1
P. REL
P. SIZE
CALCULATE ADDRESS OF NEXT HOLE
10$ PART 1
20$
CALCULATE SIZE OF LAST HOLE.
P. REL
P. SIZE
HOLE BIG ENOUGH?
N
40$ PART 1
30$
HOLE BIG ENOUGH WAS FOUND FOR REQUESTING TASK. LINK PCB INTO LIST.
P. REL
P. SUB
40$
RETURN TO CALLER. C-BIT = 1 IF HOLE NOT FOUND.

Figure 2-7 $FNDSP Logical Flow Diagram (Part 1 of 1)

2.3.6 $ICHKP Logical Flow Diagram

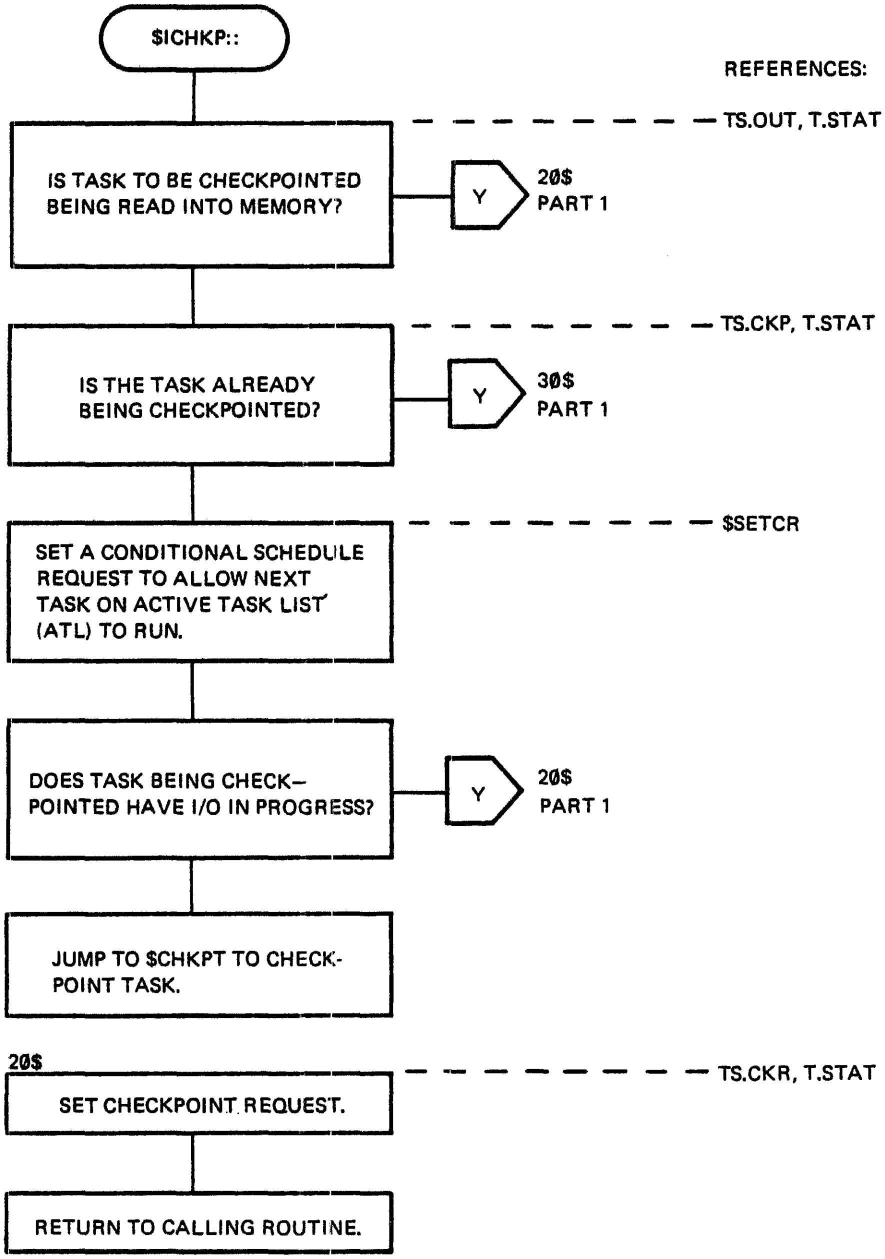

$ICHKP::
REFERENCES:
TS.OUT, T.STAT
IS TASK TO BE CHECKPOINTED BEING READ INTO MEMORY?
Y
20$ PART 1
TS.CKP, T.STAT
IS THE TASK ALREADY BEING CHECKPOINTED?
Y
30$ PART 1
$SETCR
SET A CONDITIONAL SCHEDULE REQUEST TO ALLOW NEXT TASK ON ACTIVE TASK LIST (ATL) TO RUN.
DOES TASK BEING CHECK-POINTED HAVE I/O IN PROGRESS?
Y
20$ PART 1
JUMP TO $CHKPT TO CHECK-POINT TASK.
20$
TS.CKR, T.STAT
SET CHECKPOINT REQUEST.
RETURN TO CALLING ROUTINE.

Figure 2-8 $ICHKP Logical Flow Diagram (Part 1 of 1)

2.3.7 $NXTSK Logical Flow Diagram

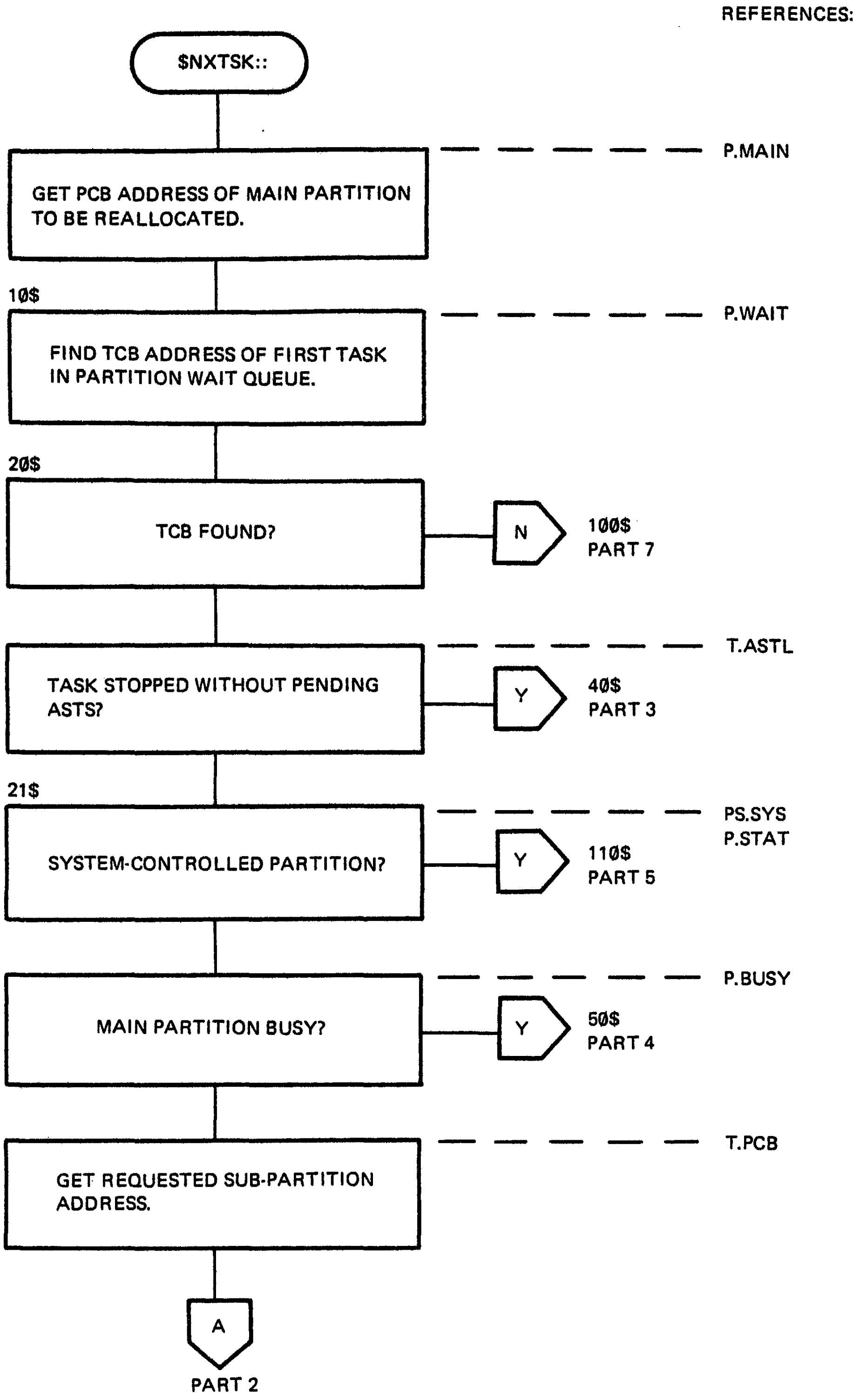

REFERENCES:
$NXTSK::
GET PCB ADDRESS OF MAIN PARTITION TO BE REALLOCATED.
P.MAIN
10$
FIND TCB ADDRESS OF FIRST TASK IN PARTITION WAIT QUEUE.
P.WAIT
20$
TCB FOUND?
N
100$ PART 7
TASK STOPPED WITHOUT PENDING ASTS?
T.ASTL
Y
40$ PART 3
21$
SYSTEM-CONTROLLED PARTITION?
PS.SYS
P.STAT
Y
110$ PART 5
MAIN PARTITION BUSY?
P.BUSY
Y
50$ PART 4
GET REQUESTED SUB-PARTITION ADDRESS.
T.PCB
A
PART 2

Figure 2-9 $NXTSK Logical Flow Diagram (Part 1 of 7)

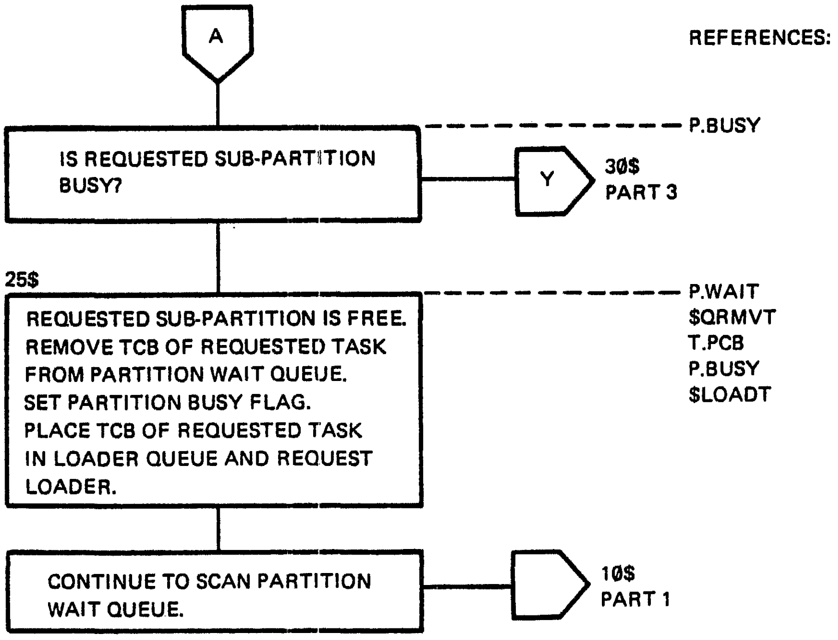

A
REFERENCES:
P.BUSY
IS REQUESTED SUB-PARTITION BUSY?
Y
3Ø$
PART 3
25$
P.WAIT
$QRMVT
T.PCB
P.BUSY
$LOADT
REQUESTED SUB-PARTITION IS FREE. REMOVE TCB OF REQUESTED TASK FROM PARTITION WAIT QUEUE. SET PARTITION BUSY FLAG. PLACE TCB OF REQUESTED TASK IN LOADER QUEUE AND REQUEST LOADER.
CONTINUE TO SCAN PARTITION WAIT QUEUE.
1Ø$
PART 1

Figure 2-9 $NXTSK Logical Flow Diagram (Part 2 of 7)

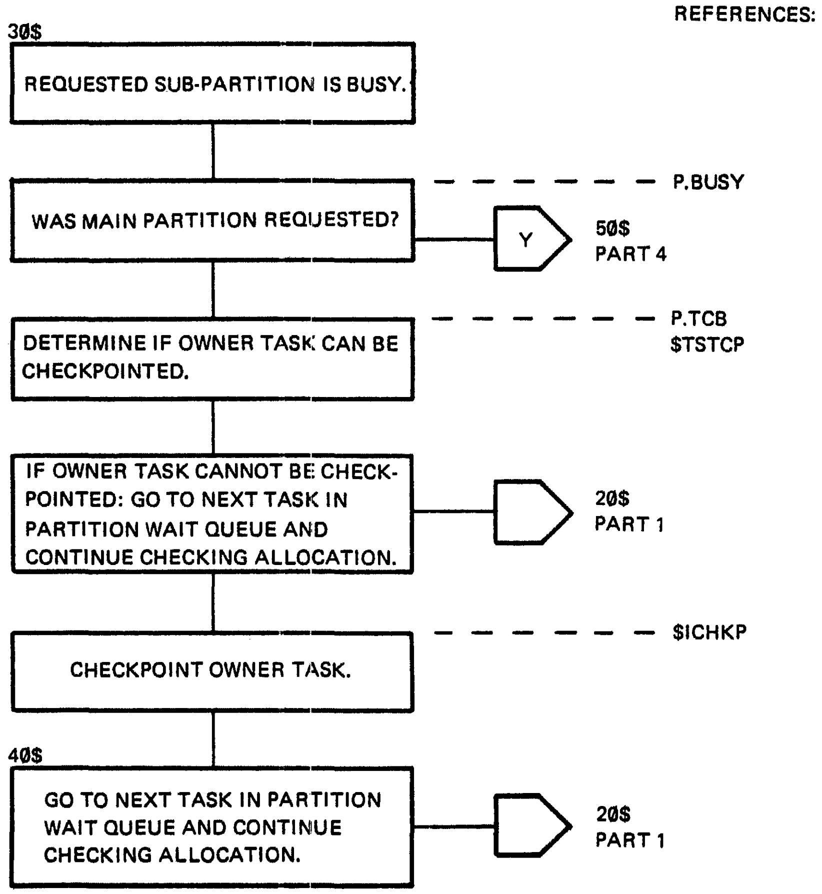

REFERENCES:
3Ø$
REQUESTED SUB-PARTITION IS BUSY.
P.BUSY
WAS MAIN PARTITION REQUESTED?
Y
5Ø$
PART 4
P.TCB
$TSTCP
DETERMINE IF OWNER TASK CAN BE CHECKPOINTED.
IF OWNER TASK CANNOT BE CHECK-POINTED: GO TO NEXT TASK IN PARTITION WAIT QUEUE AND CONTINUE CHECKING ALLOCATION.
2Ø$
PART 1
$ICHKP
CHECKPOINT OWNER TASK.
4Ø$
GO TO NEXT TASK IN PARTITION WAIT QUEUE AND CONTINUE CHECKING ALLOCATION.
2Ø$
PART 1

Figure 2-9 $NXTSK Logical Flow Diagram (Part 3 of 7)

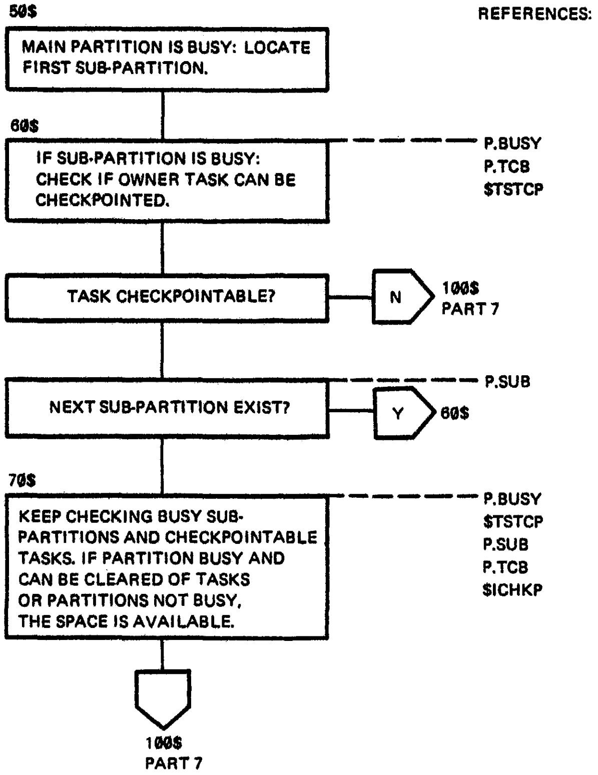

50$
MAIN PARTITION IS BUSY: LOCATE FIRST SUB-PARTITION.
60$
IF SUB-PARTITION IS BUSY: CHECK IF OWNER TASK CAN BE CHECKPOINTED.
TASK CHECKPOINTABLE?
N
100$ PART 7
NEXT SUB-PARTITION EXIST?
Y
60$
70$
KEEP CHECKING BUSY SUB-PARTITIONS AND CHECKPOINTABLE TASKS. IF PARTITION BUSY AND CAN BE CLEARED OF TASKS OR PARTITIONS NOT BUSY, THE SPACE IS AVAILABLE.
100$ PART 7
REFERENCES:
P.BUSY
P.TCB
$TSTCP
P.SUB
P.BUSY
$TSTCP
P.SUB
P.TCB
$ICHKP

Figure 2-9 $NXTSK Logical Flow Diagram (Part 4 of 7)

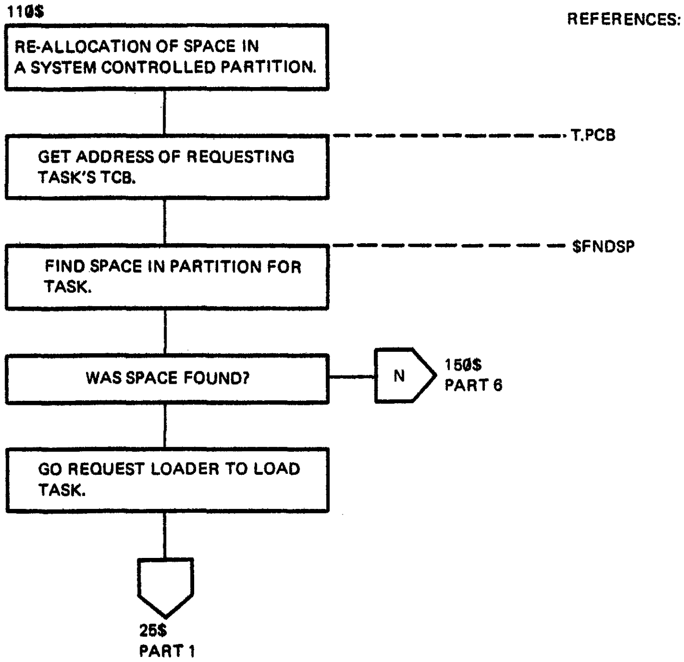

110$
RE-ALLOCATION OF SPACE IN A SYSTEM CONTROLLED PARTITION.
GET ADDRESS OF REQUESTING TASK'S TCB.
FIND SPACE IN PARTITION FOR TASK.
WAS SPACE FOUND?
N
150$ PART 6
GO REQUEST LOADER TO LOAD TASK.
25$ PART 1
REFERENCES:
T.PCB
$FNDSP

Figure 2-9 $NXTSK Logical Flow Diagram (Part 5 of 7)

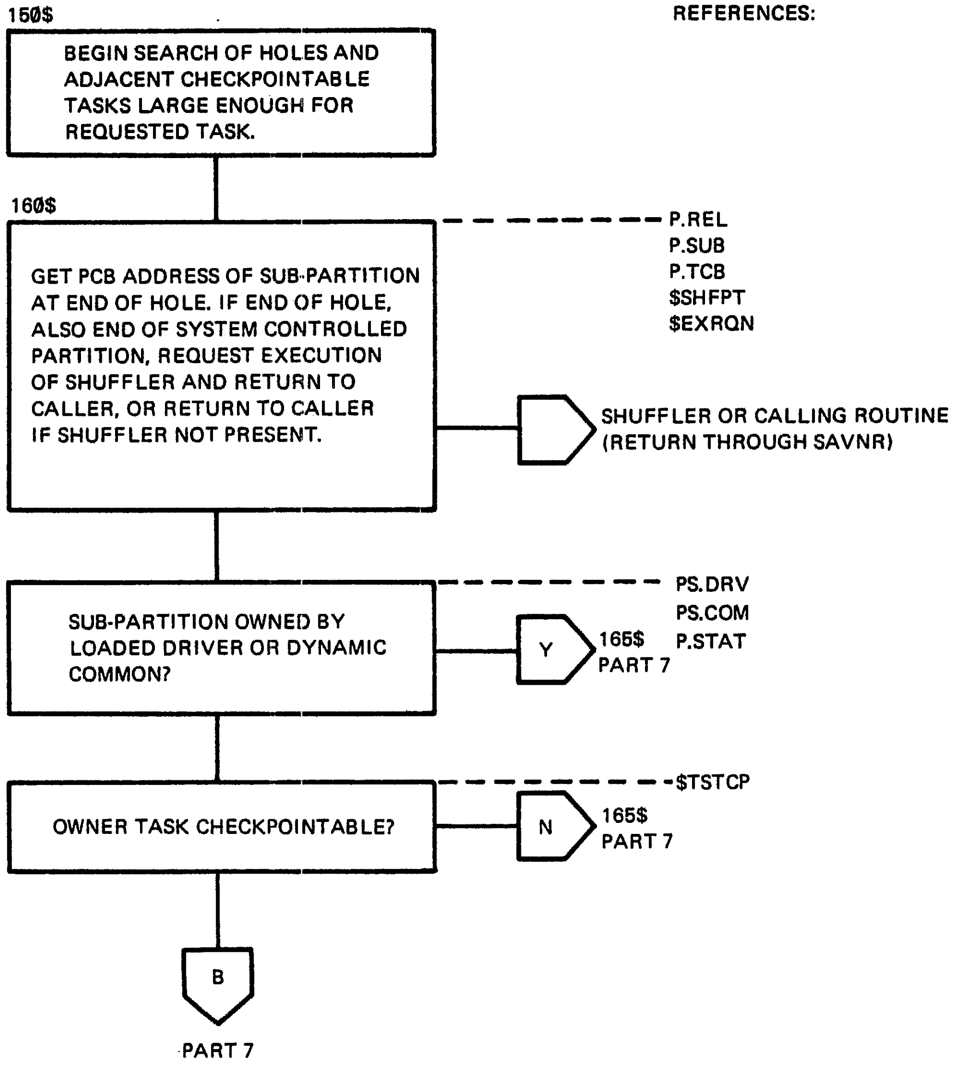

150$
REFERENCES:
BEGIN SEARCH OF HOLES AND ADJACENT CHECKPOINTABLE TASKS LARGE ENOUGH FOR REQUESTED TASK.
160$
GET PCB ADDRESS OF SUB-PARTITION AT END OF HOLE. IF END OF HOLE, ALSO END OF SYSTEM CONTROLLED PARTITION, REQUEST EXECUTION OF SHUFFLER AND RETURN TO CALLER, OR RETURN TO CALLER IF SHUFFLER NOT PRESENT.
P.REL
P.SUB
P.TCB
$SHFPT
$EXRQN
SHUFFLER OR CALLING ROUTINE (RETURN THROUGH SAVNR)
SUB-PARTITION OWNED BY LOADED DRIVER OR DYNAMIC COMMON?
PS.DRV
PS.COM
P.STAT
Y
165$
PART 7
OWNER TASK CHECKPOINTABLE?
$TSTCP
N
165$
PART 7
B
PART 7

Figure 2-9 $NXTSK Logical Flow Diagram (Part 6 of 7)

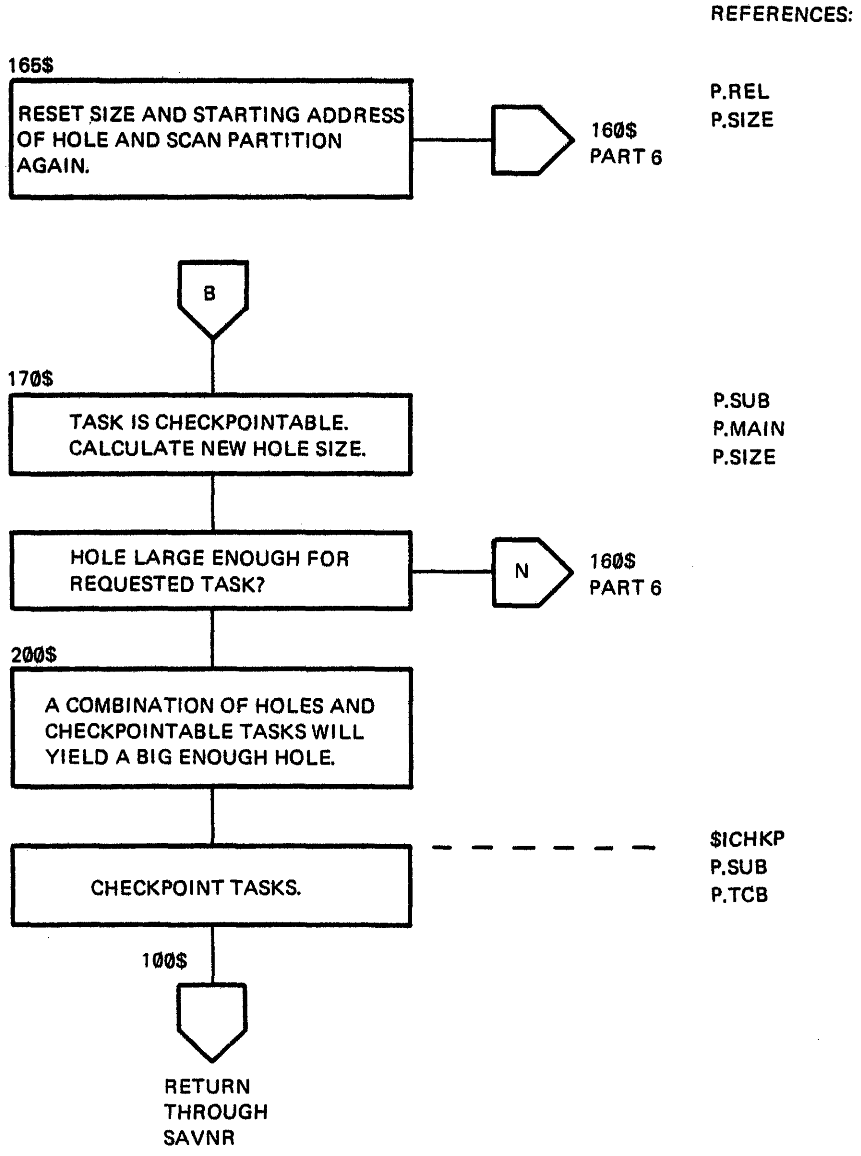

REFERENCES:
165$
RESET SIZE AND STARTING ADDRESS OF HOLE AND SCAN PARTITION AGAIN.
160$ PART 6
P.REL
P.SIZE
B
170$
TASK IS CHECKPOINTABLE. CALCULATE NEW HOLE SIZE.
P.SUB
P.MAIN
P.SIZE
HOLE LARGE ENOUGH FOR REQUESTED TASK?
N
160$ PART 6
200$
A COMBINATION OF HOLES AND CHECKPOINTABLE TASKS WILL YIELD A BIG ENOUGH HOLE.
CHECKPOINT TASKS.
$ICHKP
P.SUB
P.TCB
100$
RETURN THROUGH SAVNR

Figure 2-9 $NXTSK Logical Flow Diagram (Part 7 of 7)

2.3.8 $TSTCP Logical Flow Diagram

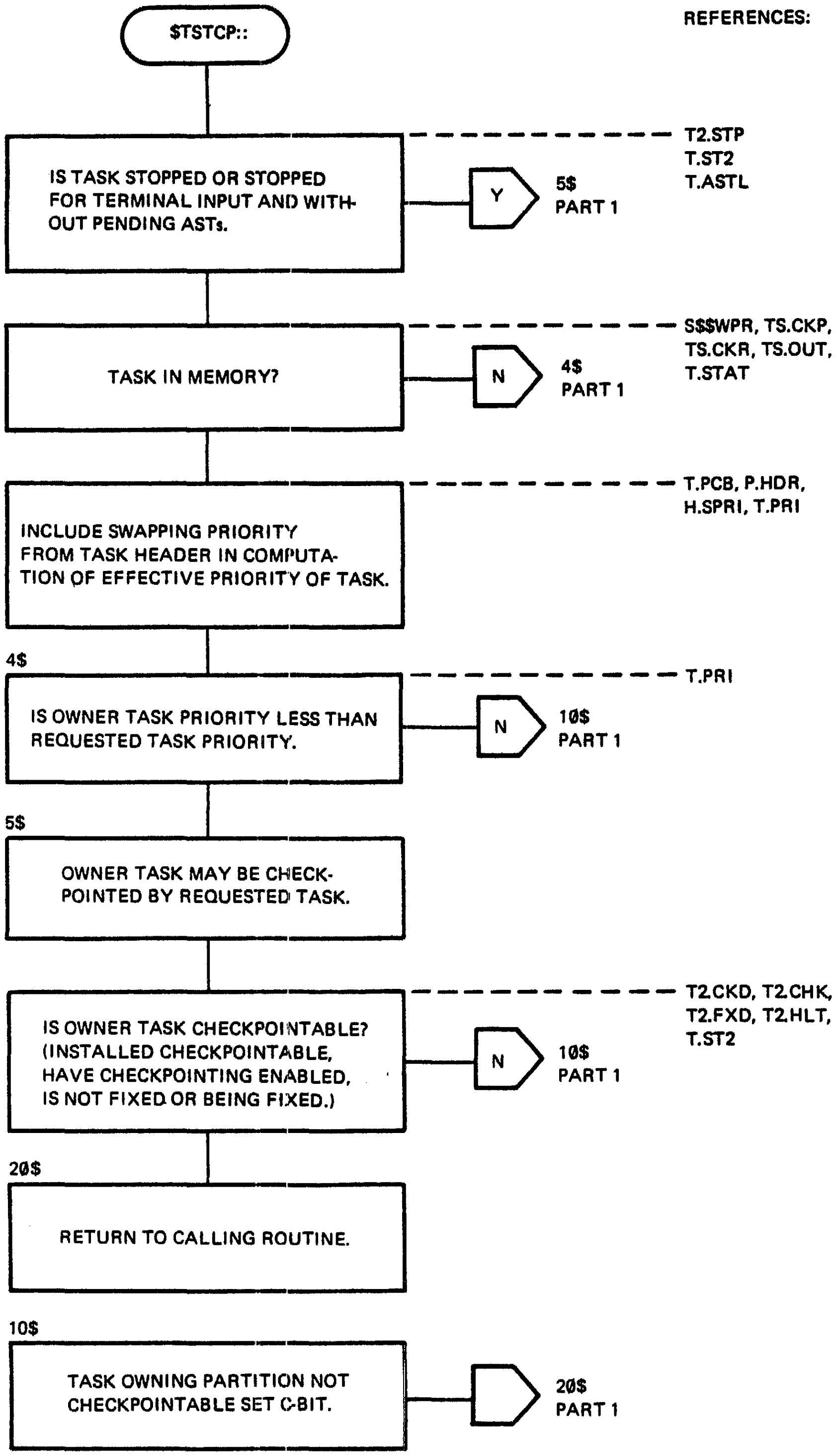

$TSTCP::
REFERENCES:
IS TASK STOPPED OR STOPPED FOR TERMINAL INPUT AND WITHOUT PENDING ASTs.
Y
5$ PART 1
T2.STP
T.ST2
T.ASTL
TASK IN MEMORY?
N
4$ PART 1
S$$WPR, TS.CKP,
TS.CKR, TS.OUT,
T.STAT
INCLUDE SWAPPING PRIORITY FROM TASK HEADER IN COMPUTATION OF EFFECTIVE PRIORITY OF TASK.
T.PCB, P.HDR,
H.SPRI, T.PRI
4$
IS OWNER TASK PRIORITY LESS THAN REQUESTED TASK PRIORITY.
N
10$ PART 1
T.PRI
5$
OWNER TASK MAY BE CHECKPOINTED BY REQUESTED TASK.
IS OWNER TASK CHECKPOINTABLE? (INSTALLED CHECKPOINTABLE, HAVE CHECKPOINTING ENABLED, IS NOT FIXED OR BEING FIXED.)
N
10$ PART 1
T2.CKD, T2.CHK,
T2.FXD, T2.HLT,
T.ST2
20$
RETURN TO CALLING ROUTINE.
10$
TASK OWNING PARTITION NOT CHECKPOINTABLE SET C-BIT.
20$ PART 1

Figure 2-10 $TSTCP Logical Flow Diagram (Part 1 of 1)

2.3.9 Loader Logical Flow Diagram

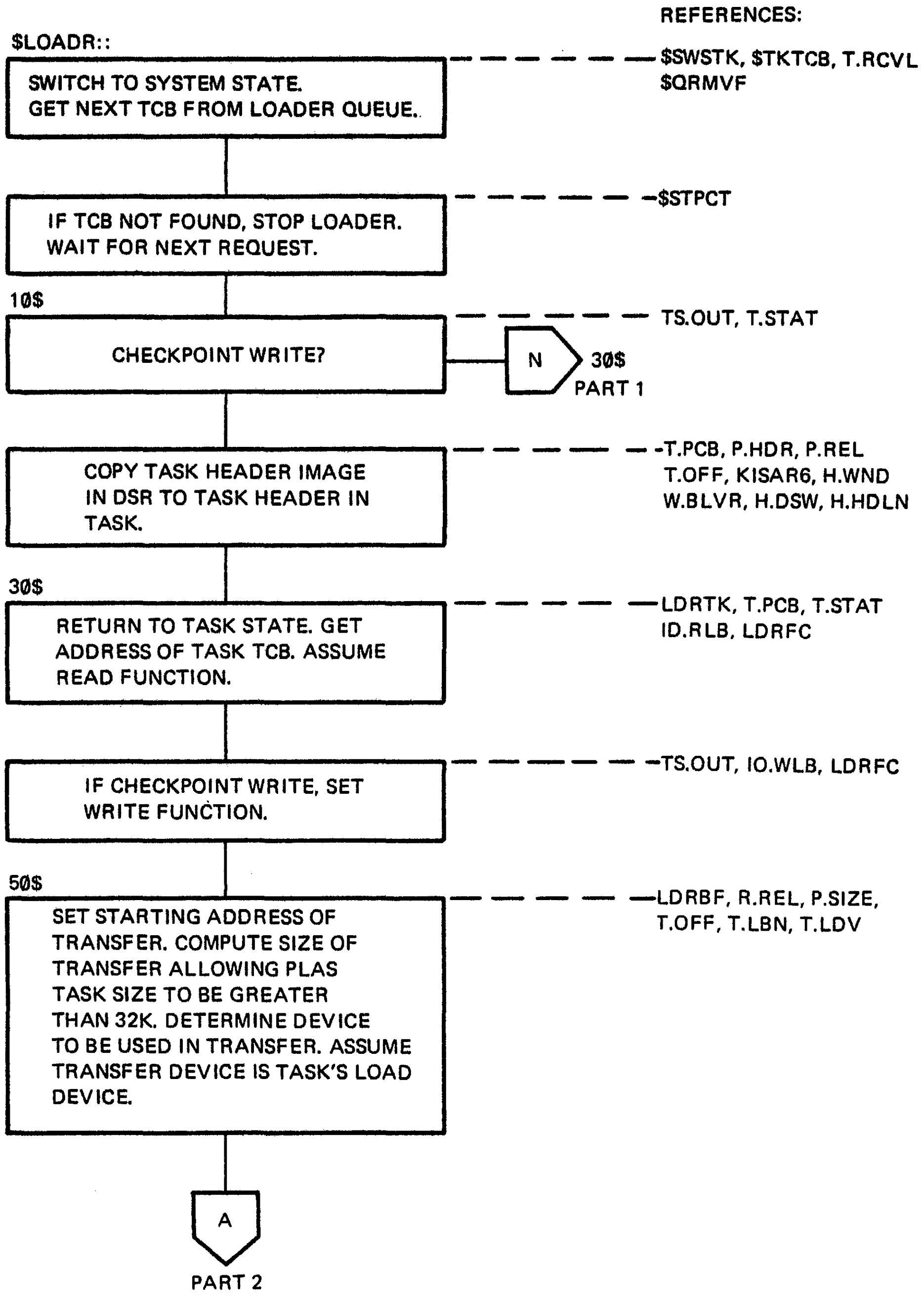

REFERENCES:
$LOADR::
SWITCH TO SYSTEM STATE. GET NEXT TCB FROM LOADER QUEUE..
$SWSTK, $TKTCB, T.RCVL $QRMVF
IF TCB NOT FOUND, STOP LOADER. WAIT FOR NEXT REQUEST.
$STPCT
10$
CHECKPOINT WRITE?
TS.OUT, T.STAT
N
30$ PART 1
COPY TASK HEADER IMAGE IN DSR TO TASK HEADER IN TASK.
T.PCB, P.HDR, P.REL T.OFF, KISAR6, H.WND W.BLVR, H.DSW, H.HDLN
30$
RETURN TO TASK STATE. GET ADDRESS OF TASK TCB. ASSUME READ FUNCTION.
LDRTK, T.PCB, T.STAT ID.RLB, LDRFC
IF CHECKPOINT WRITE, SET WRITE FUNCTION.
TS.OUT, IO.WLB, LDRFC
50$
SET STARTING ADDRESS OF TRANSFER. COMPUTE SIZE OF TRANSFER ALLOWING PLAS TASK SIZE TO BE GREATER THAN 32K. DETERMINE DEVICE TO BE USED IN TRANSFER. ASSUME TRANSFER DEVICE IS TASK'S LOAD DEVICE.
LDRBF, R.REL, P.SIZE, T.OFF, T.LBN, T.LDV
A
PART 2

Figure 2-11 Loader Logical Flow Diagram (Part 1 of 8)

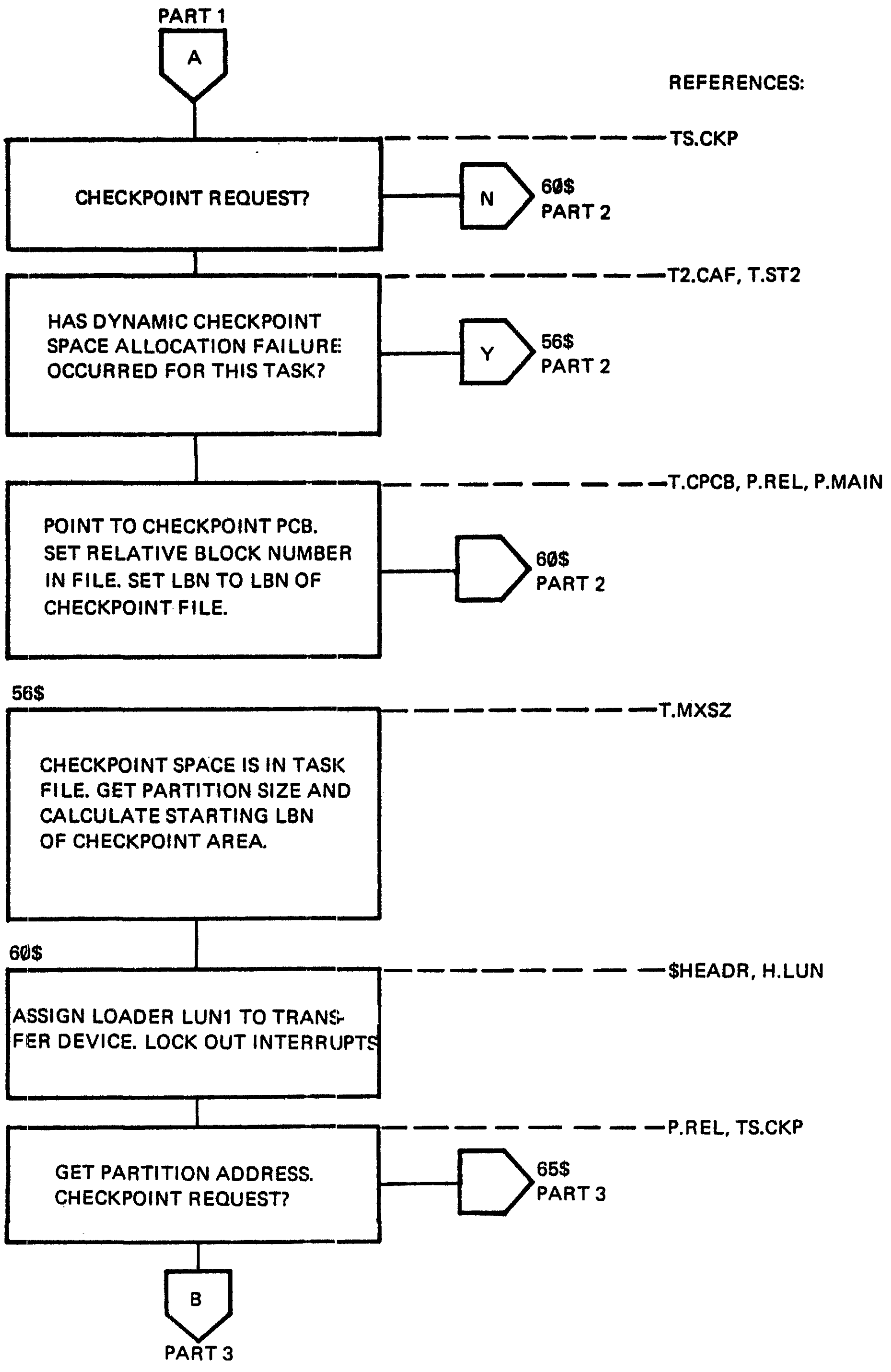

PART 1
A
REFERENCES:
TS.CKP
CHECKPOINT REQUEST?
N
60$
PART 2
T2.CAF, T.ST2
HAS DYNAMIC CHECKPOINT SPACE ALLOCATION FAILURE OCCURRED FOR THIS TASK?
Y
56$
PART 2
T.CPCB, P.REL, P.MAIN
POINT TO CHECKPOINT PCB. SET RELATIVE BLOCK NUMBER IN FILE. SET LBN TO LBN OF CHECKPOINT FILE.
60$
PART 2
56$
T.MXSZ
CHECKPOINT SPACE IS IN TASK FILE. GET PARTITION SIZE AND CALCULATE STARTING LBN OF CHECKPOINT AREA.
60$
$HEADR, H.LUN
ASSIGN LOADER LUN1 TO TRANSFER DEVICE. LOCK OUT INTERRUPTS
P.REL, TS.CKP
GET PARTITION ADDRESS. CHECKPOINT REQUEST?
65$
PART 3
B
PART 3

Figure 2-11 Loader Logical Flow Diagram (Part 2 of 8)

MEMORY RESOURCE ALLOCATION

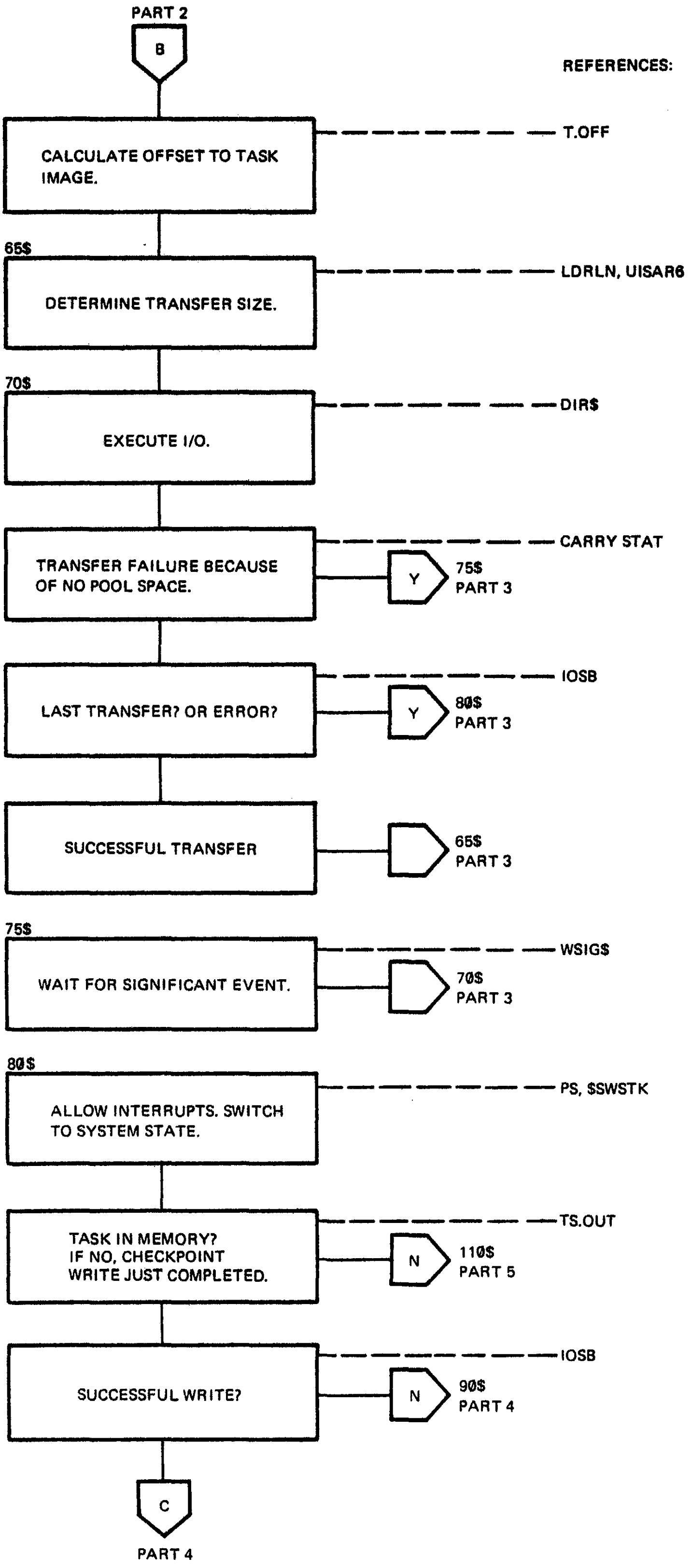


Figure 2-11 Loader Logical Flow Diagram (Part 3 of 8)

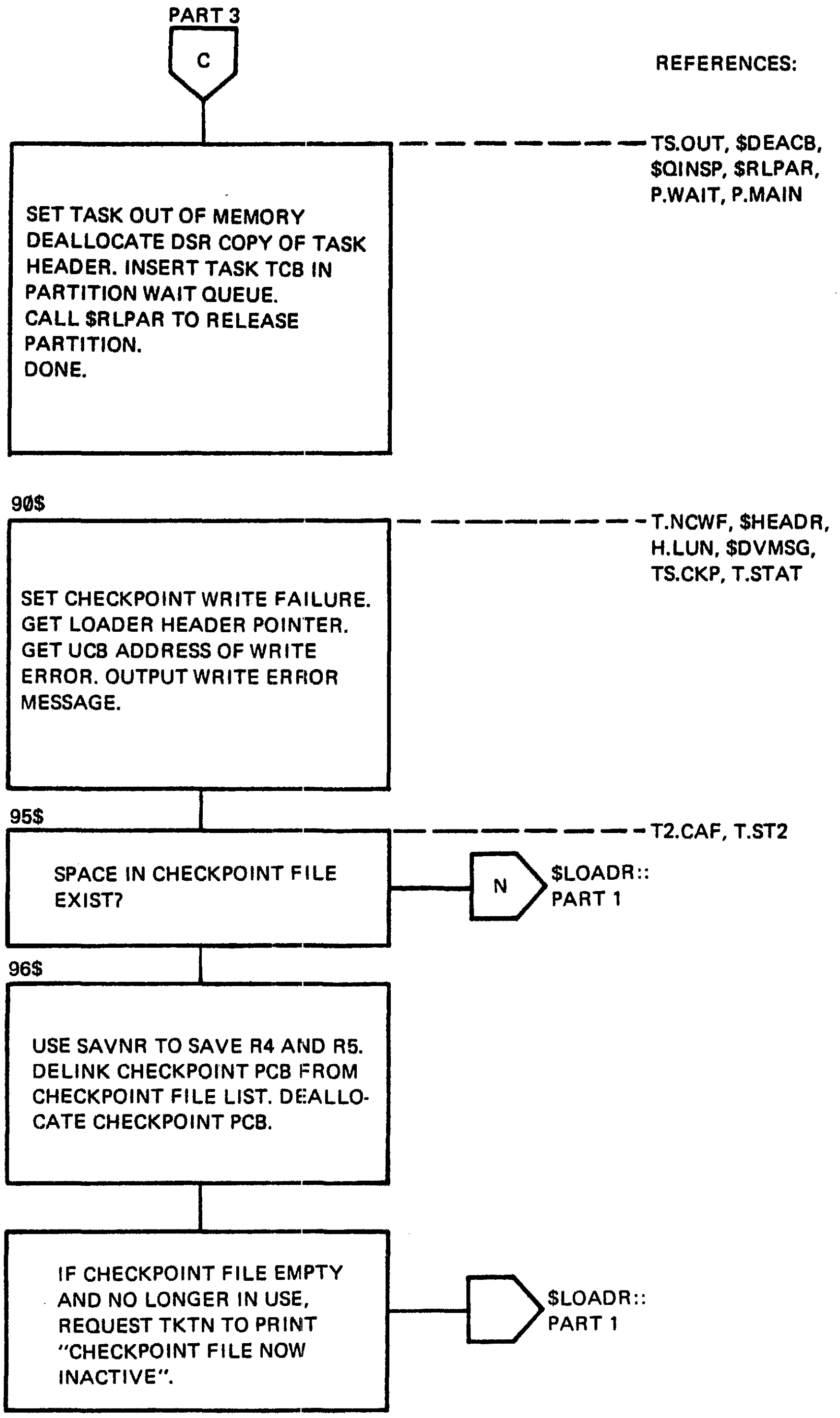

PART 3
C
REFERENCES:
TS.OUT, $DEACB, $QINSP, $RLPAR, P.WAIT, P.MAIN
SET TASK OUT OF MEMORY DEALLOCATE DSR COPY OF TASK HEADER. INSERT TASK TCB IN PARTITION WAIT QUEUE. CALL $RLPAR TO RELEASE PARTITION. DONE.
90$
T.NCWF, $HEADR, H.LUN, $DVMSG, TS.CKP, T.STAT
SET CHECKPOINT WRITE FAILURE. GET LOADER HEADER POINTER. GET UCB ADDRESS OF WRITE ERROR. OUTPUT WRITE ERROR MESSAGE.
95$
T2.CAF, T.ST2
SPACE IN CHECKPOINT FILE EXIST?
N
$LOADR:: PART 1
96$
USE SAVNR TO SAVE R4 AND R5. DELINK CHECKPOINT PCB FROM CHECKPOINT FILE LIST. DEALLOCATE CHECKPOINT PCB.
IF CHECKPOINT FILE EMPTY AND NO LONGER IN USE, REQUEST TKTN TO PRINT "CHECKPOINT FILE NOW INACTIVE".
$LOADR:: PART 1

Figure 2-11 Loader Logical Flow Diagram (Part 4 of 8)

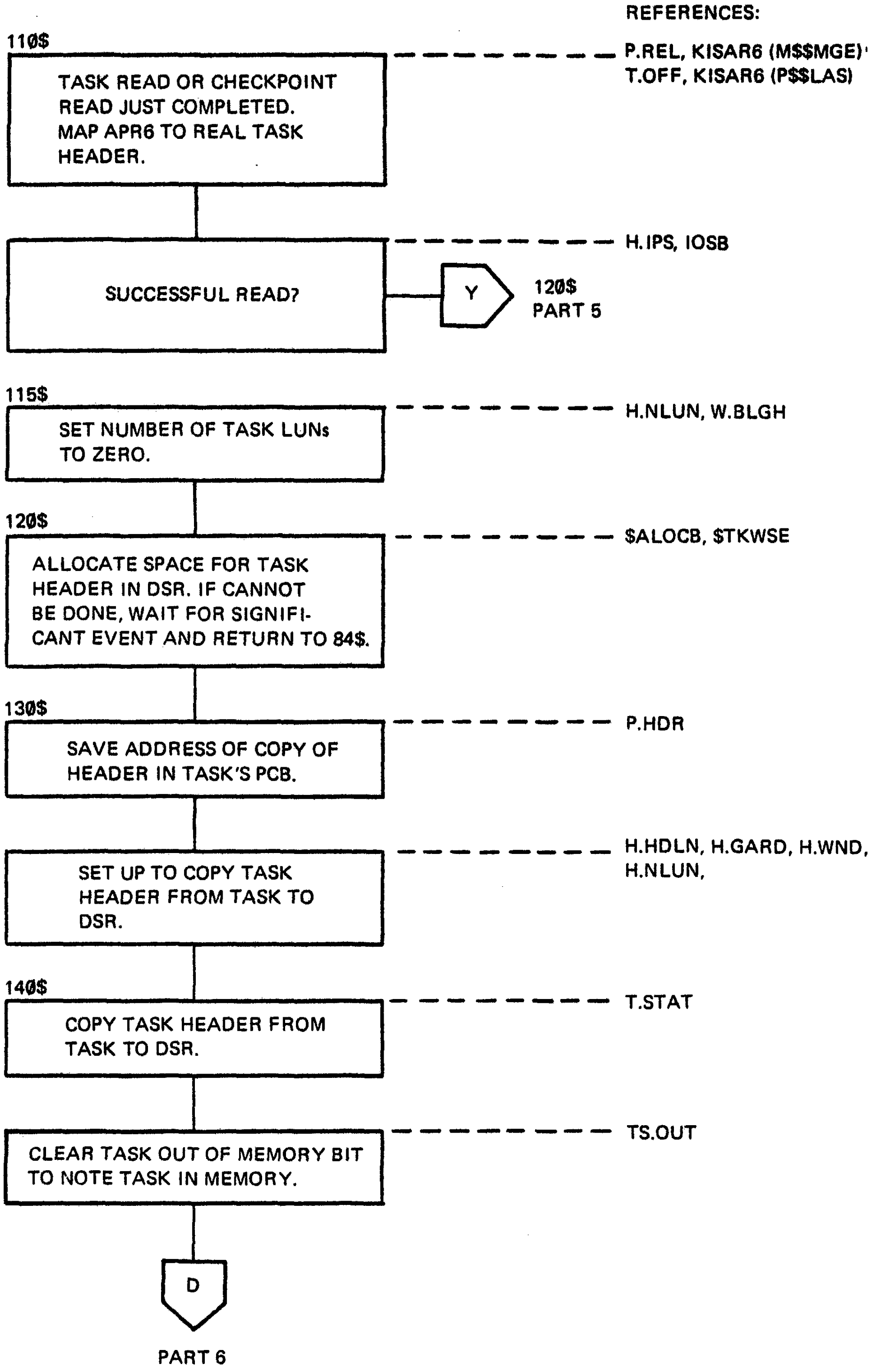

REFERENCES:
110$
TASK READ OR CHECKPOINT READ JUST COMPLETED. MAP APR6 TO REAL TASK HEADER.
P.REL, KISAR6 (M$$MGE)'
T.OFF, KISAR6 (P$$LAS)
SUCCESSFUL READ?
H.IPS, IOSB
Y
120$
PART 5
115$
SET NUMBER OF TASK LUNs TO ZERO.
H.NLUN, W.BLGH
120$
ALLOCATE SPACE FOR TASK HEADER IN DSR. IF CANNOT BE DONE, WAIT FOR SIGNIFICANT EVENT AND RETURN TO 84$.
$ALOCB, $TKWSE
130$
SAVE ADDRESS OF COPY OF HEADER IN TASK'S PCB.
P.HDR
SET UP TO COPY TASK HEADER FROM TASK TO DSR.
H.HDLN, H.GARD, H.WND, H.NLUN,
140$
COPY TASK HEADER FROM TASK TO DSR.
T.STAT
CLEAR TASK OUT OF MEMORY BIT TO NOTE TASK IN MEMORY.
TS.OUT
D
PART 6

Figure 2-11 Loader Logical Flow Diagram (Part 5 of 8)

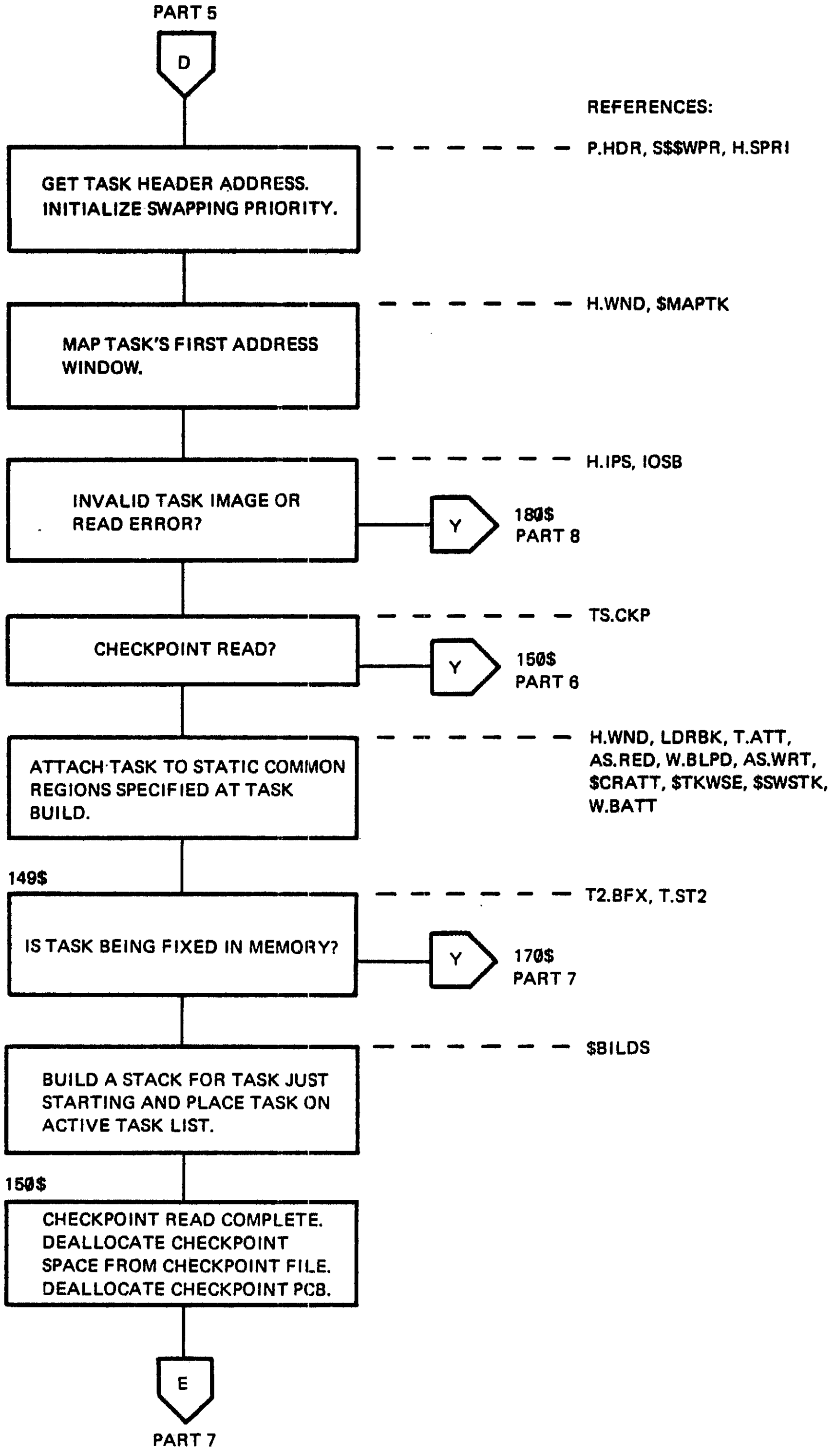

PART 5
D
REFERENCES:
P.HDR, S$$WPR, H.SPRI
GET TASK HEADER ADDRESS. INITIALIZE SWAPPING PRIORITY.
H.WND, $MAPTK
MAP TASK'S FIRST ADDRESS WINDOW.
H.IPS, IOSB
INVALID TASK IMAGE OR READ ERROR?
Y
180$
PART 8
TS.CKP
CHECKPOINT READ?
Y
150$
PART 6
H.WND, LDRBK, T.ATT, AS.RED, W.BLPD, AS.WRT, $CRATT, $TKWSE, $SWSTK, W.BATT
ATTACH TASK TO STATIC COMMON REGIONS SPECIFIED AT TASK BUILD.
149$
T2.BFX, T.ST2
IS TASK BEING FIXED IN MEMORY?
Y
170$
PART 7
$BILDS
BUILD A STACK FOR TASK JUST STARTING AND PLACE TASK ON ACTIVE TASK LIST.
150$
CHECKPOINT READ COMPLETE. DEALLOCATE CHECKPOINT SPACE FROM CHECKPOINT FILE. DEALLOCATE CHECKPOINT PCB.
E
PART 7

Figure 2-11 Loader Logical Flow Diagram (Part 6 of 8)

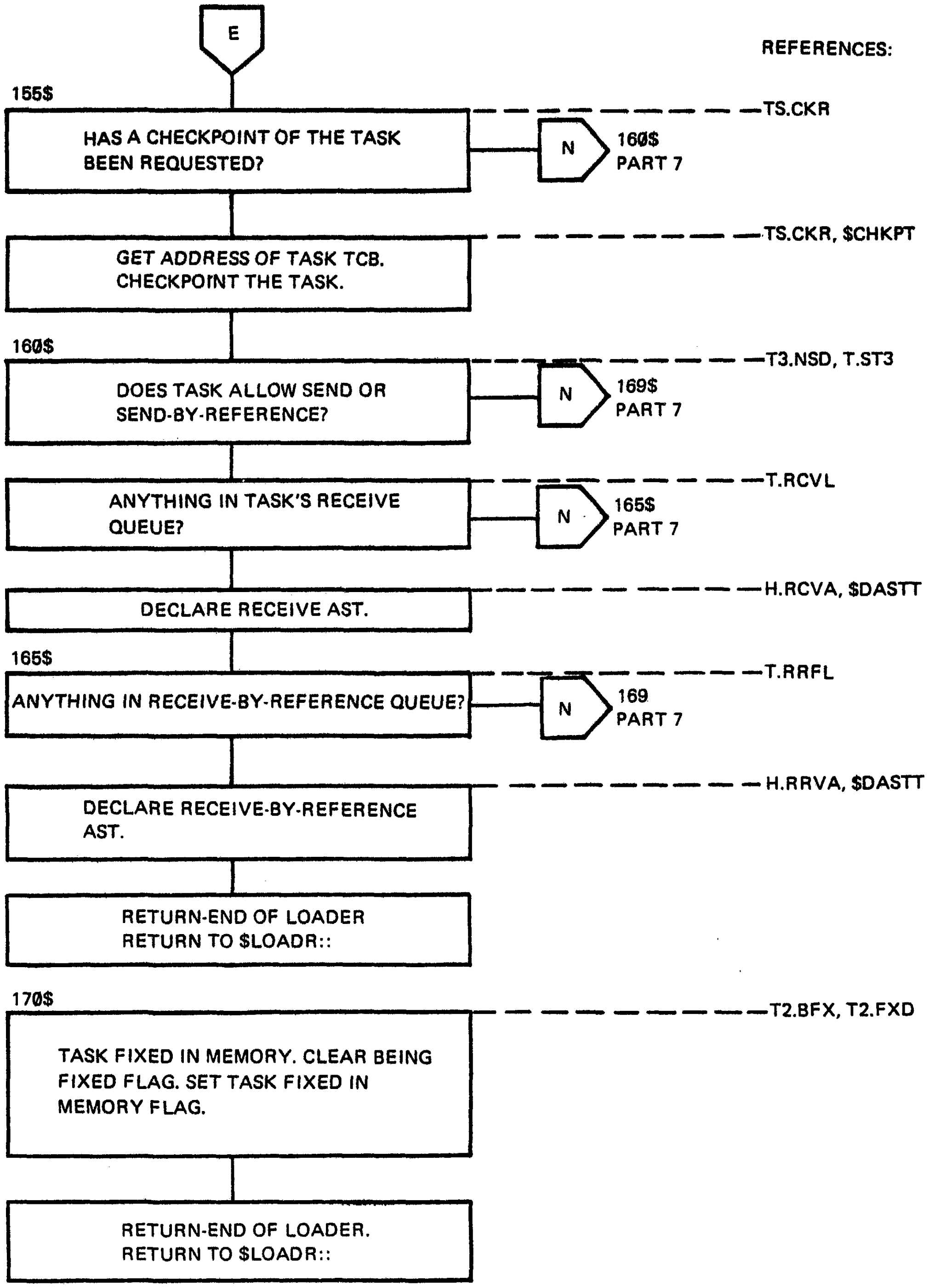

E
REFERENCES:
155$
HAS A CHECKPOINT OF THE TASK BEEN REQUESTED?
TS.CKR
N
160$ PART 7
GET ADDRESS OF TASK TCB. CHECKPOINT THE TASK.
TS.CKR, $CHKPT
160$
DOES TASK ALLOW SEND OR SEND-BY-REFERENCE?
T3.NSD, T.ST3
N
169$ PART 7
ANYTHING IN TASK'S RECEIVE QUEUE?
T.RCVL
N
165$ PART 7
DECLARE RECEIVE AST.
H.RCVA, $DASTT
165$
ANYTHING IN RECEIVE-BY-REFERENCE QUEUE?
T.RRFL
N
169 PART 7
DECLARE RECEIVE-BY-REFERENCE AST.
H.RRVA, $DASTT
RETURN-END OF LOADER RETURN TO $LOADR::
170$
TASK FIXED IN MEMORY. CLEAR BEING FIXED FLAG. SET TASK FIXED IN MEMORY FLAG.
T2.BFX, T2.FXD
RETURN-END OF LOADER. RETURN TO $LOADR::

Figure 2-11 Loader Logical Flow Diagram (Part 7 of 8)

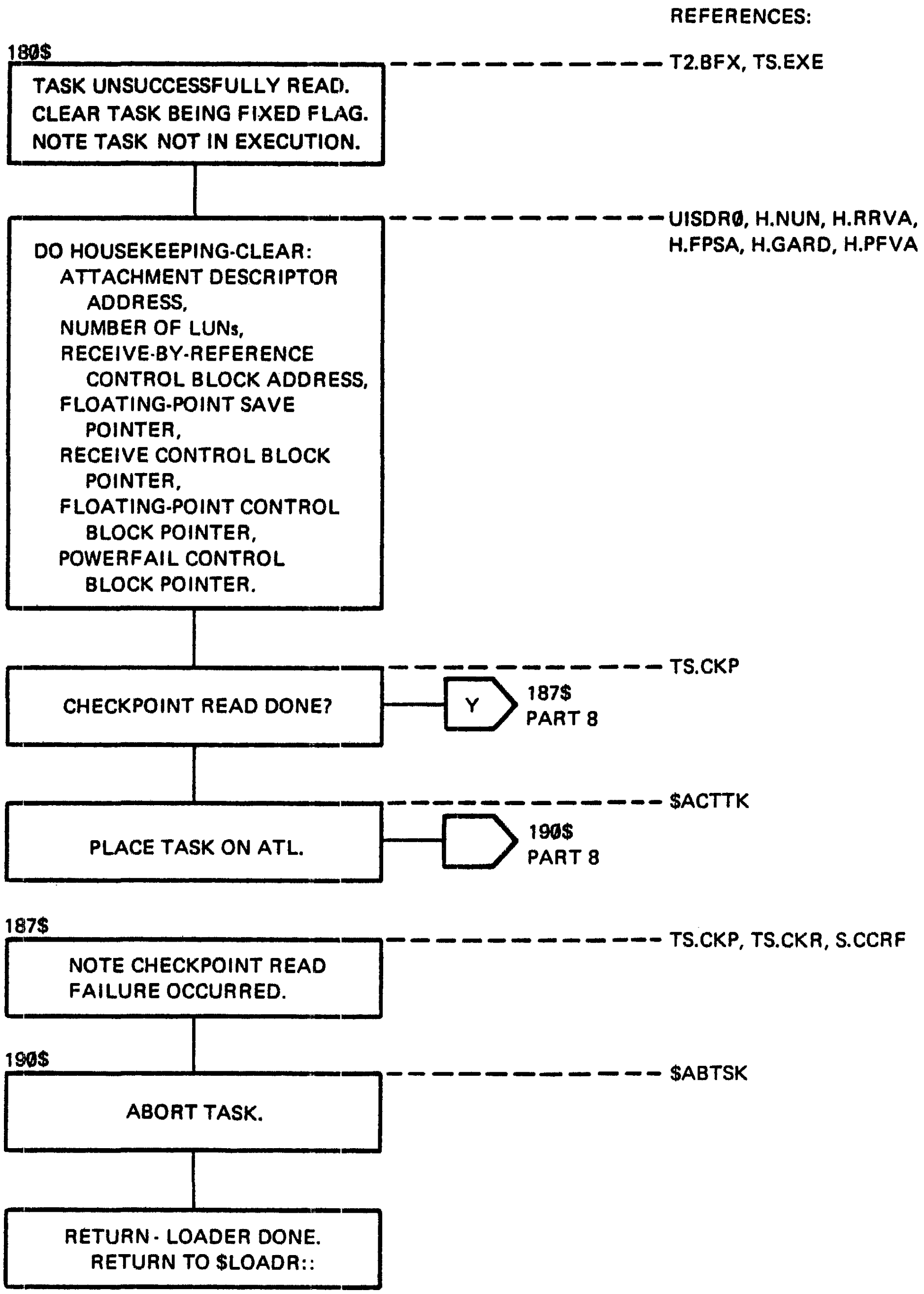

REFERENCES:
180$
TASK UNSUCCESSFULLY READ.
CLEAR TASK BEING FIXED FLAG.
NOTE TASK NOT IN EXECUTION.
T2.BFX, TS.EXE
DO HOUSEKEEPING-CLEAR:
ATTACHMENT DESCRIPTOR ADDRESS,
NUMBER OF LUNs,
RECEIVE-BY-REFERENCE CONTROL BLOCK ADDRESS,
FLOATING-POINT SAVE POINTER,
RECEIVE CONTROL BLOCK POINTER,
FLOATING-POINT CONTROL BLOCK POINTER,
POWERFAIL CONTROL BLOCK POINTER.
UISDR0, H.NUN, H.RRVA, H.FPSA, H.GARD, H.PFVA
CHECKPOINT READ DONE?
Y
187$
PART 8
TS.CKP
PLACE TASK ON ATL.
190$
PART 8
$ACTTK
187$
NOTE CHECKPOINT READ FAILURE OCCURRED.
TS.CKP, TS.CKR, S.CCRF
190$
ABORT TASK.
$ABTSK
RETURN - LOADER DONE.
RETURN TO $LOADR::

Figure 2-11 Loader Logical Flow Diagram (Part 8 of 8)

2.3.10 Shuffler Logical Flow Diagram

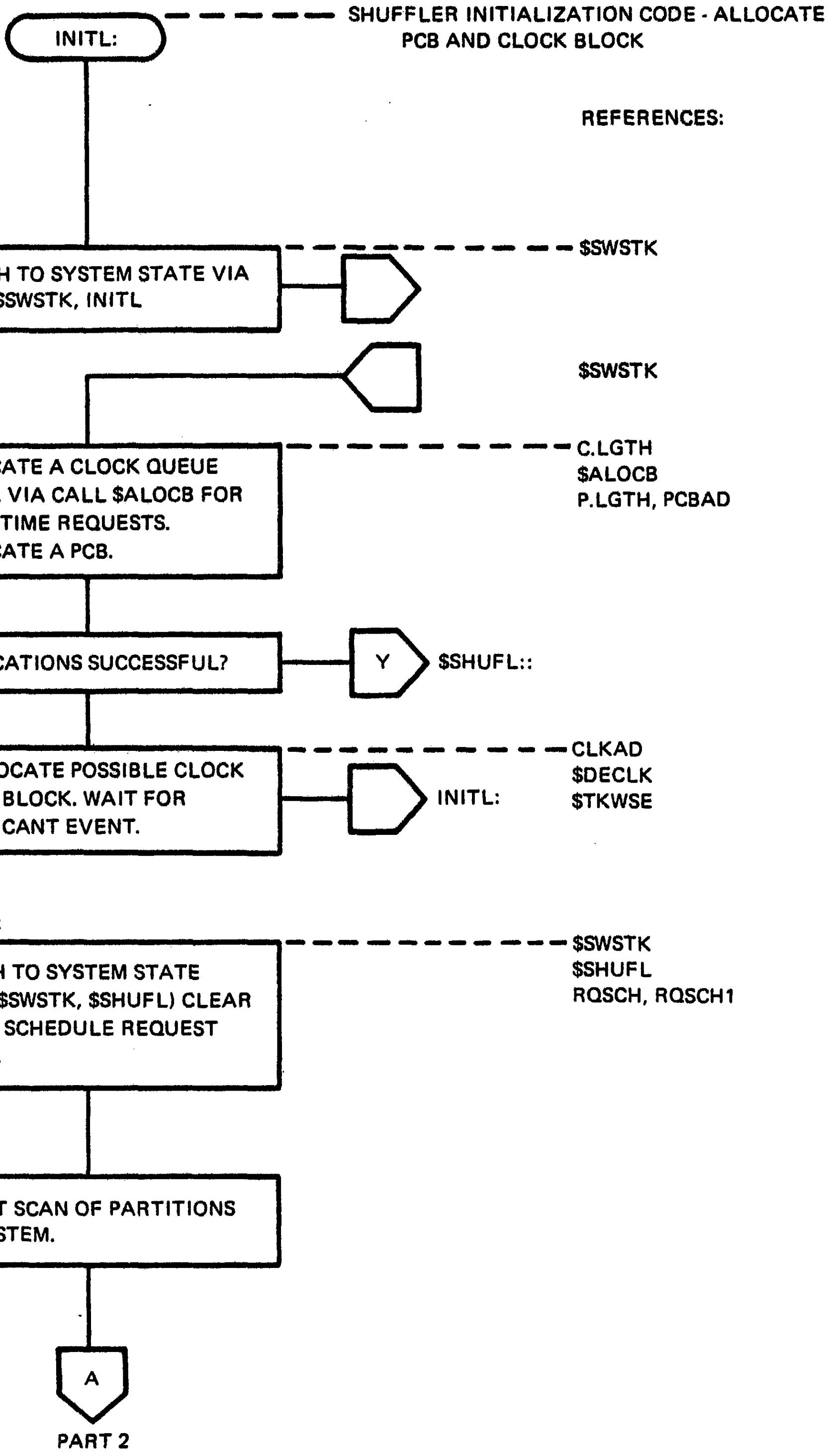

INITL:
SHUFFLER INITIALIZATION CODE - ALLOCATE PCB AND CLOCK BLOCK
REFERENCES:
SWITCH TO SYSTEM STATE VIA CALL $SWSTK, INITL
$SWSTK
$SWSTK
ALLOCATE A CLOCK QUEUE BLOCK VIA CALL $ALOCB FOR MARK TIME REQUESTS. ALLOCATE A PCB.
C.LGTH
$ALOCB
P.LGTH, PCBAD
ALLOCATIONS SUCCESSFUL?
Y
$SHUFL::
DEALLOCATE POSSIBLE CLOCK QUEUE BLOCK. WAIT FOR SIGNIFICANT EVENT.
INITL:
CLKAD
$DECLK
$TKWSE
$SHUFL::
SWITCH TO SYSTEM STATE (CALL $SWSTK, $SHUFL) CLEAR LOCAL SCHEDULE REQUEST FLAGS.
$SWSTK
$SHUFL
RQSCH, RQSCH1
START SCAN OF PARTITIONS IN SYSTEM.
A
PART 2

Figure 2-12 Shuffler Logical Flow Diagram (Part 1 of 11)

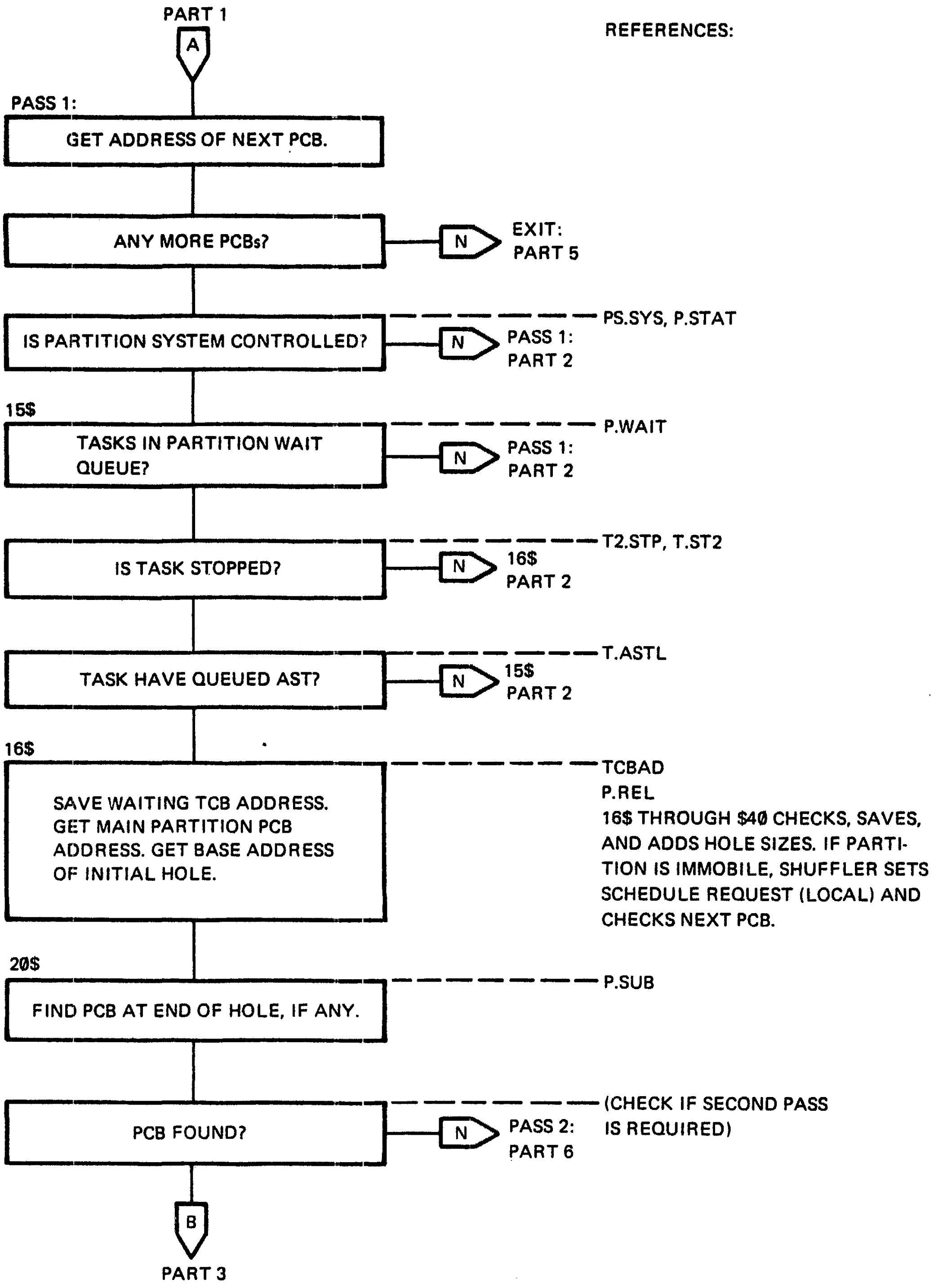

PART 1
A
PASS 1:
GET ADDRESS OF NEXT PCB.
ANY MORE PCBs?
N
EXIT:
PART 5
IS PARTITION SYSTEM CONTROLLED?
N
PASS 1:
PART 2
15$
TASKS IN PARTITION WAIT QUEUE?
N
PASS 1:
PART 2
IS TASK STOPPED?
N
16$
PART 2
TASK HAVE QUEUED AST?
N
15$
PART 2
16$
SAVE WAITING TCB ADDRESS. GET MAIN PARTITION PCB ADDRESS. GET BASE ADDRESS OF INITIAL HOLE.
20$
FIND PCB AT END OF HOLE, IF ANY.
PCB FOUND?
N
PASS 2:
PART 6
B
PART 3
REFERENCES:
PS.SYS, P.STAT
P.WAIT
T2.STP, T.ST2
T.ASTL
TCBAD
P.REL
16$ THROUGH $40 CHECKS, SAVES, AND ADDS HOLE SIZES. IF PARTITION IS IMMOBILE, SHUFFLER SETS SCHEDULE REQUEST (LOCAL) AND CHECKS NEXT PCB.
P.SUB
(CHECK IF SECOND PASS IS REQUIRED)

Figure 2-12 Shufller Logical Flow Diagram (Part 2 of 11)

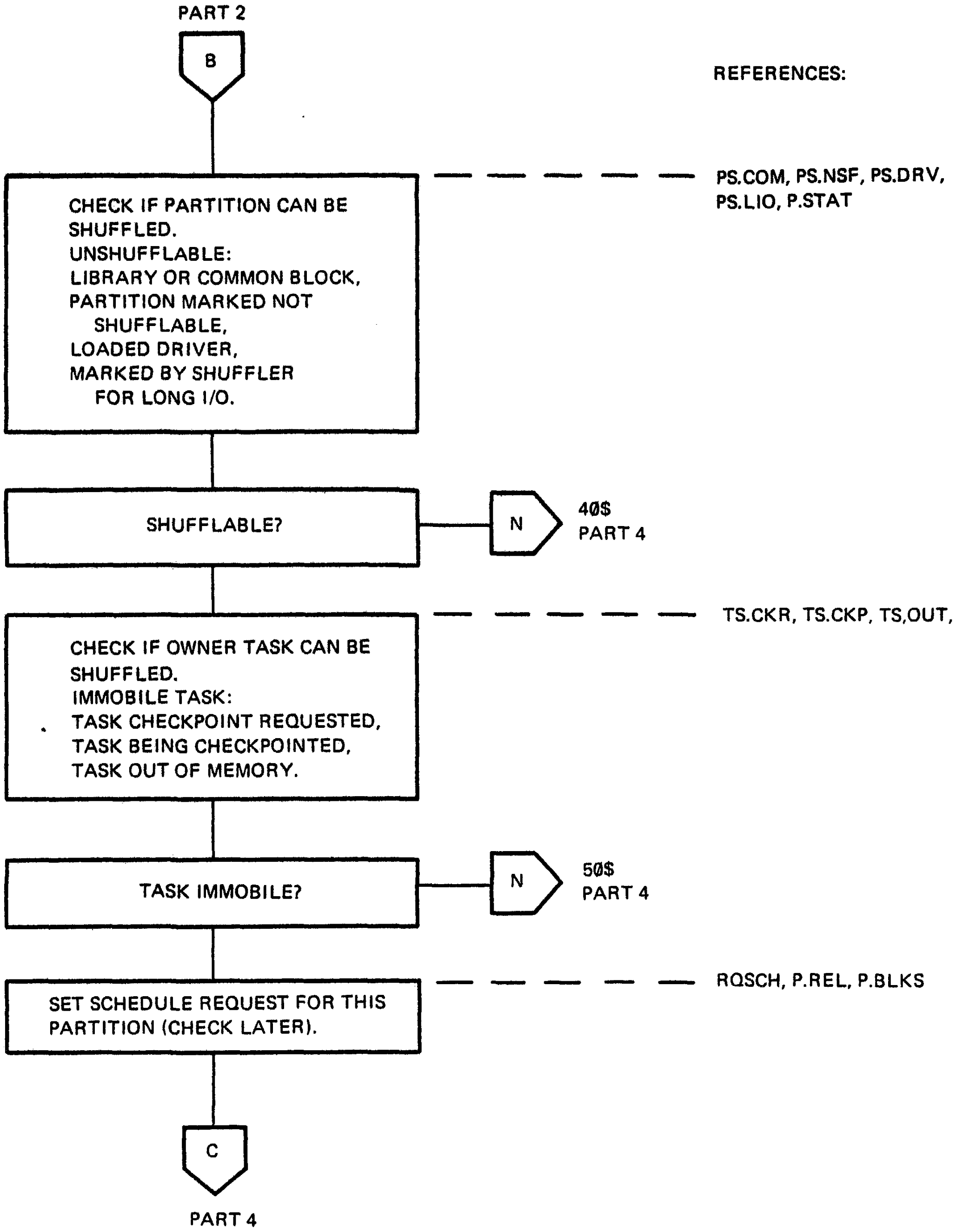

PART 2
B
REFERENCES:
CHECK IF PARTITION CAN BE SHUFFLED.
UNSHUFFLABLE:
LIBRARY OR COMMON BLOCK,
PARTITION MARKED NOT SHUFFLABLE,
LOADED DRIVER,
MARKED BY SHUFFLER FOR LONG I/O.
PS.COM, PS.NSF, PS.DRV, PS.LIO, P.STAT
SHUFFLABLE?
N
40$ PART 4
CHECK IF OWNER TASK CAN BE SHUFFLED.
IMMOBILE TASK:
TASK CHECKPOINT REQUESTED,
TASK BEING CHECKPOINTED,
TASK OUT OF MEMORY.
TS.CKR, TS.CKP, TS,OUT,
TASK IMMOBILE?
N
50$ PART 4
SET SCHEDULE REQUEST FOR THIS PARTITION (CHECK LATER).
RQSCH, P.REL, P.BLKS
C
PART 4

Figure 2-12 Shuffler Logical Flow Diagram (Part 3 of 11)

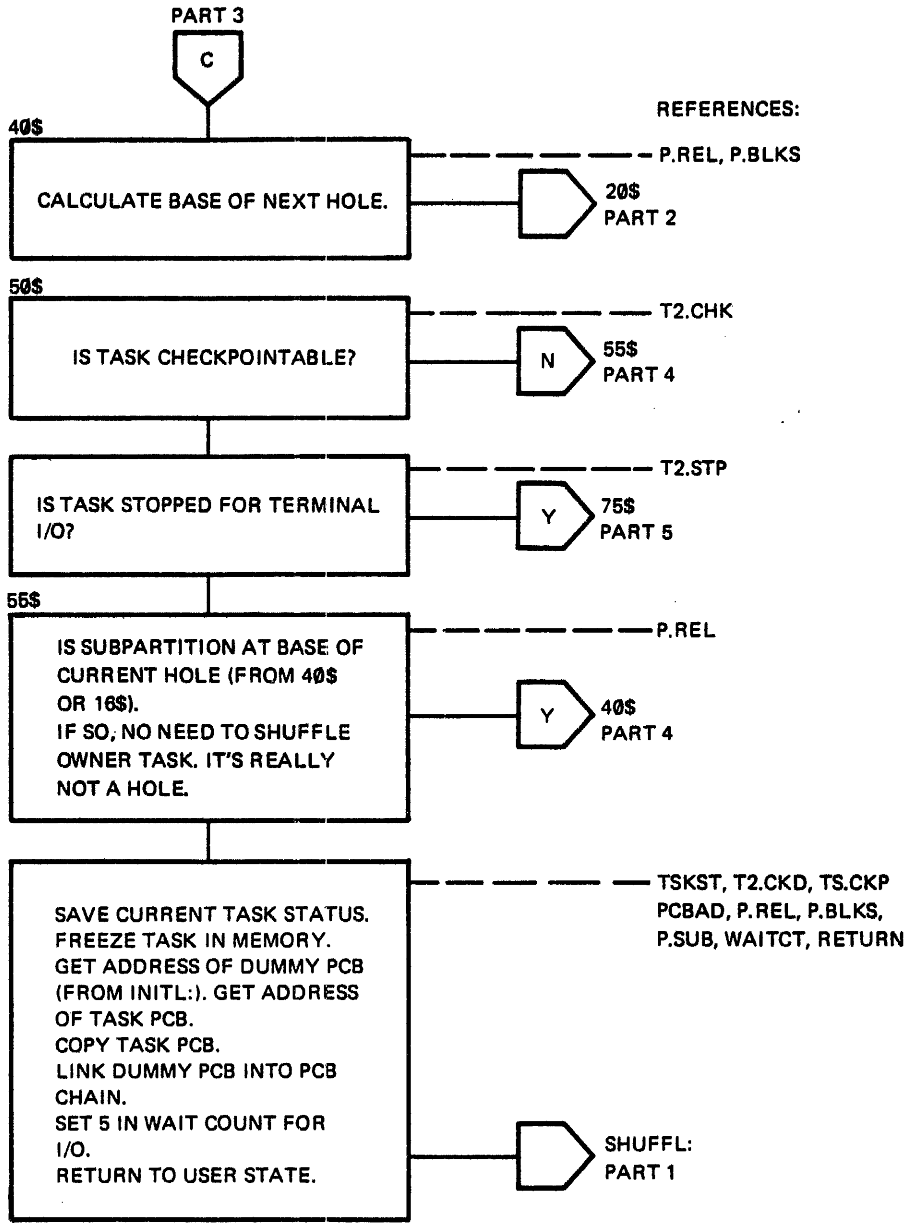

PART 3
C
REFERENCES:
40$
CALCULATE BASE OF NEXT HOLE.
P.REL, P.BLKS
20$
PART 2
50$
IS TASK CHECKPOINTABLE?
T2.CHK
N
55$
PART 4
IS TASK STOPPED FOR TERMINAL I/O?
T2.STP
Y
75$
PART 5
55$
IS SUBPARTITION AT BASE OF CURRENT HOLE (FROM 40$ OR 16$).
IF SO, NO NEED TO SHUFFLE OWNER TASK. IT'S REALLY NOT A HOLE.
P.REL
Y
40$
PART 4
SAVE CURRENT TASK STATUS.
FREEZE TASK IN MEMORY.
GET ADDRESS OF DUMMY PCB (FROM INITL:). GET ADDRESS OF TASK PCB.
COPY TASK PCB.
LINK DUMMY PCB INTO PCB CHAIN.
SET 5 IN WAIT COUNT FOR I/O.
RETURN TO USER STATE.
TSKST, T2.CKD, TS.CKP
PCBAD, P.REL, P.BLKS,
P.SUB, WAITCT, RETURN
SHUFFL:
PART 1

Figure 2-12 Shuffler Logical Flow Diagram (Part 4 of 11)

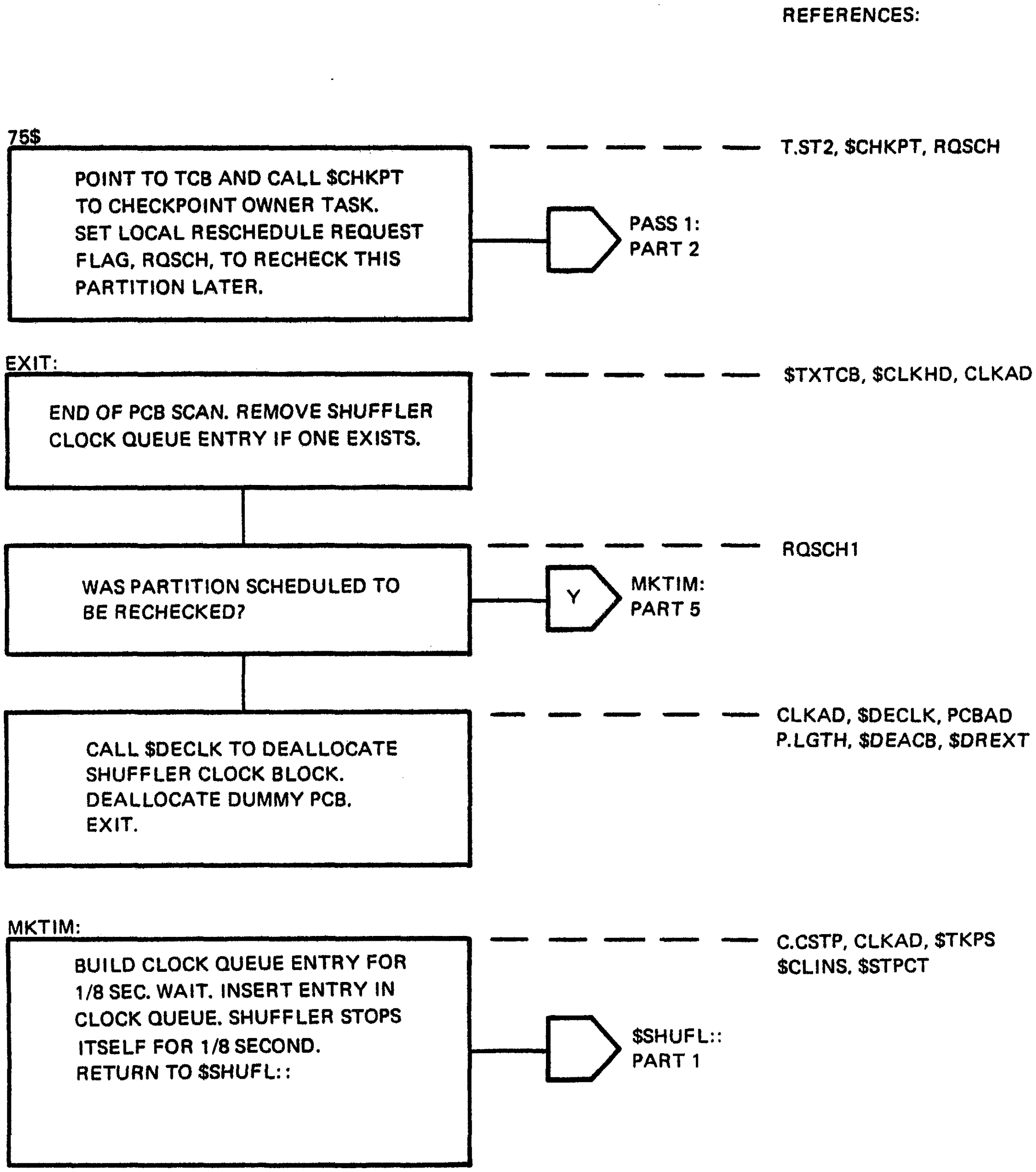
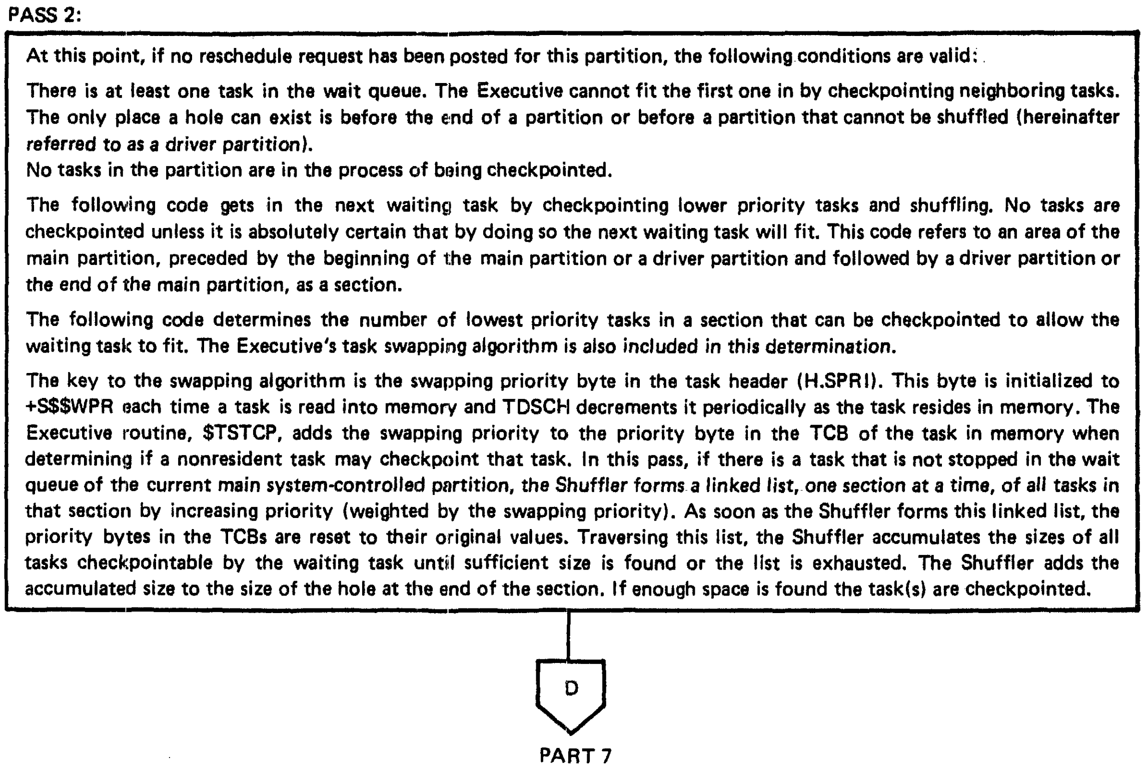

REFERENCES:
75$
POINT TO TCB AND CALL $CHKPT TO CHECKPOINT OWNER TASK. SET LOCAL RESCHEDULE REQUEST FLAG, RQSCH, TO RECHECK THIS PARTITION LATER.
T.ST2, $CHKPT, RQSCH
PASS 1: PART 2
EXIT:
END OF PCB SCAN. REMOVE SHUFFLER CLOCK QUEUE ENTRY IF ONE EXISTS.
$TXTCB, $CLKHD, CLKAD
WAS PARTITION SCHEDULED TO BE RECHECKED?
RQSCH1
Y
MKTIM: PART 5
CALL $DECLK TO DEALLOCATE SHUFFLER CLOCK BLOCK. DEALLOCATE DUMMY PCB. EXIT.
CLKAD, $DECLK, PCBAD
P.LGTH, $DEACB, $DREXT
MKTIM:
BUILD CLOCK QUEUE ENTRY FOR 1/8 SEC. WAIT. INSERT ENTRY IN CLOCK QUEUE. SHUFFLER STOPS ITSELF FOR 1/8 SECOND. RETURN TO $SHUFL::
C.CSTP, CLKAD, $TKPS
$CLINS, $STPCT
$SHUFL:: PART 1

Figure 2-12 Shuffler Logical Flow Diagram (Part 5 of 11)



PASS 2:

At this point, if no reschedule request has been posted for this partition, the following conditions are valid:

There is at least one task in the wait queue. The Executive cannot fit the first one in by checkpointing neighboring tasks. The only place a hole can exist is before the end of a partition or before a partition that cannot be shuffled (hereinafter referred to as a driver partition).
No tasks in the partition are in the process of being checkpointed.

The following code gets in the next waiting task by checkpointing lower priority tasks and shuffling. No tasks are checkpointed unless it is absolutely certain that by doing so the next waiting task will fit. This code refers to an area of the main partition, preceded by the beginning of the main partition or a driver partition and followed by a driver partition or the end of the main partition, as a section.

The following code determines the number of lowest priority tasks in a section that can be checkpointed to allow the waiting task to fit. The Executive's task swapping algorithm is also included in this determination.

The key to the swapping algorithm is the swapping priority byte in the task header (H.SPRI). This byte is initialized to +S$$WPR each time a task is read into memory and TDSCH decrements it periodically as the task resides in memory. The Executive routine, $TSTCP, adds the swapping priority to the priority byte in the TCB of the task in memory when determining if a nonresident task may checkpoint that task. In this pass, if there is a task that is not stopped in the wait queue of the current main system-controlled partition, the Shuffler forms a linked list, one section at a time, of all tasks in that section by increasing priority (weighted by the swapping priority). As soon as the Shuffler forms this linked list, the priority bytes in the TCBs are reset to their original values. Traversing this list, the Shuffler accumulates the sizes of all tasks checkpointable by the waiting task until sufficient size is found or the list is exhausted. The Shuffler adds the accumulated size to the size of the hole at the end of the section. If enough space is found the task(s) are checkpointed.

Figure 2-12 Shuffler Logical Flow Diagram (Part 6 of 11)

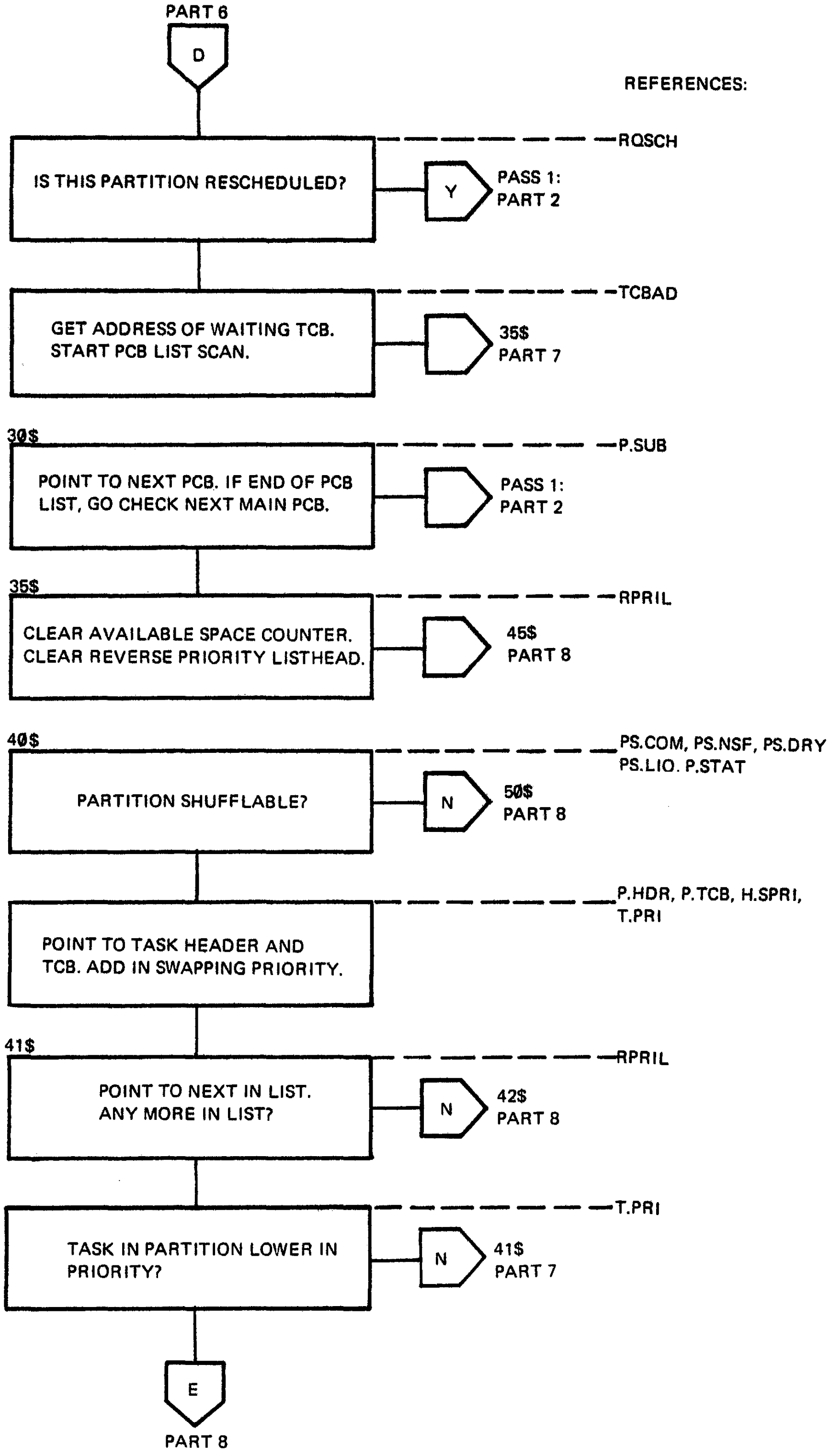


Figure 2-12 Shuffler Logical Flow Diagram (Part 7 of 11)

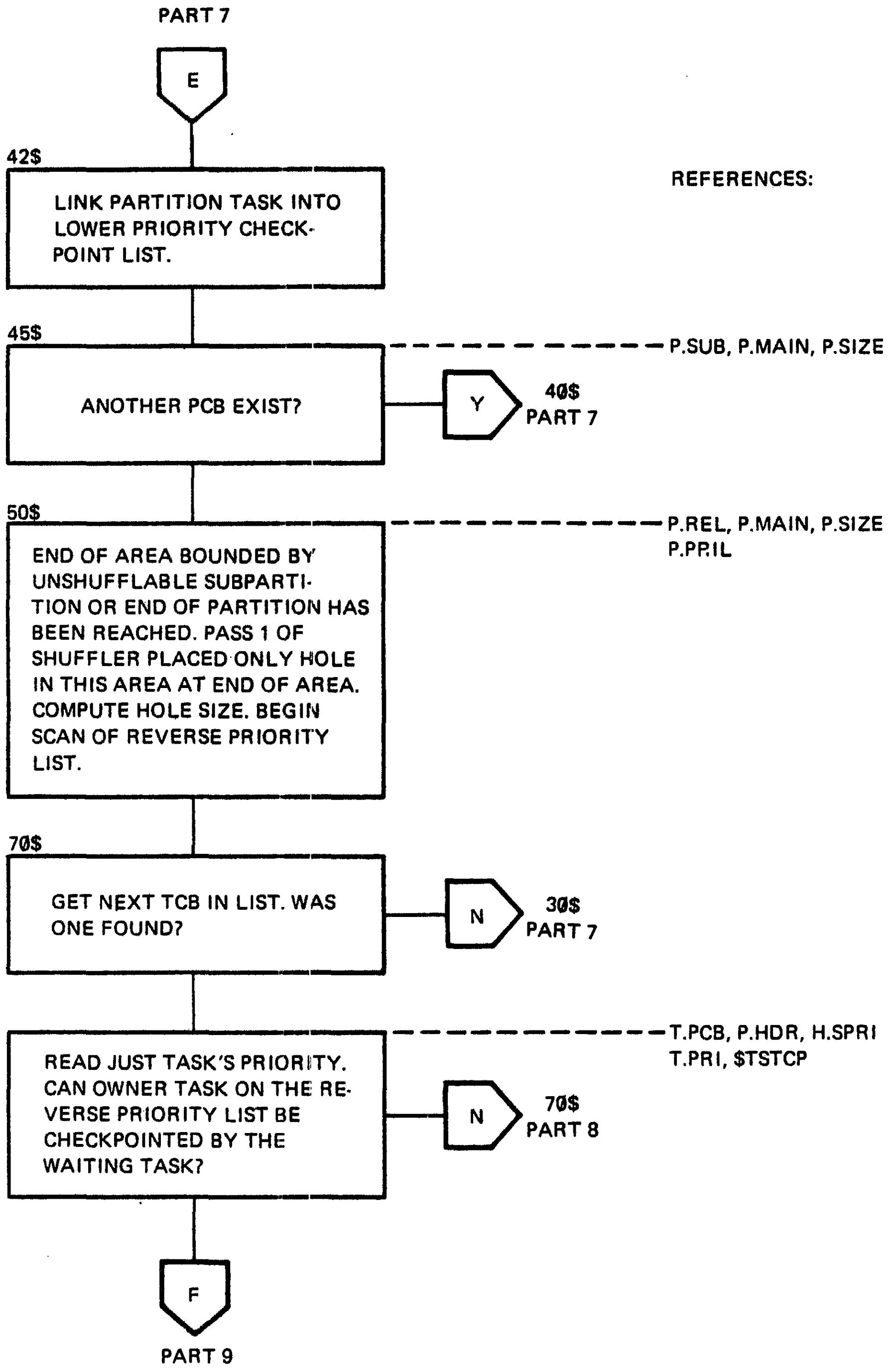

PART 7
E
42$
LINK PARTITION TASK INTO LOWER PRIORITY CHECK-POINT LIST.
REFERENCES:
45$
P.SUB, P.MAIN, P.SIZE
ANOTHER PCB EXIST?
Y
40$ PART 7
50$
P.REL, P.MAIN, P.SIZE P.PRIL
END OF AREA BOUNDED BY UNSHUFFLABLE SUBPARTI-TION OR END OF PARTITION HAS BEEN REACHED. PASS 1 OF SHUFFLER PLACED ONLY HOLE IN THIS AREA AT END OF AREA. COMPUTE HOLE SIZE. BEGIN SCAN OF REVERSE PRIORITY LIST.
70$
GET NEXT TCB IN LIST. WAS ONE FOUND?
N
30$ PART 7
T.PCB, P.HDR, H.SPRI T.PRI, $TSTCP
READ JUST TASK'S PRIORITY. CAN OWNER TASK ON THE RE-VERSE PRIORITY LIST BE CHECKPOINTED BY THE WAITING TASK?
N
70$ PART 8
F
PART 9

Figure 2-12 Shuffler Logical Flow Diagram (Part 8 of 11)

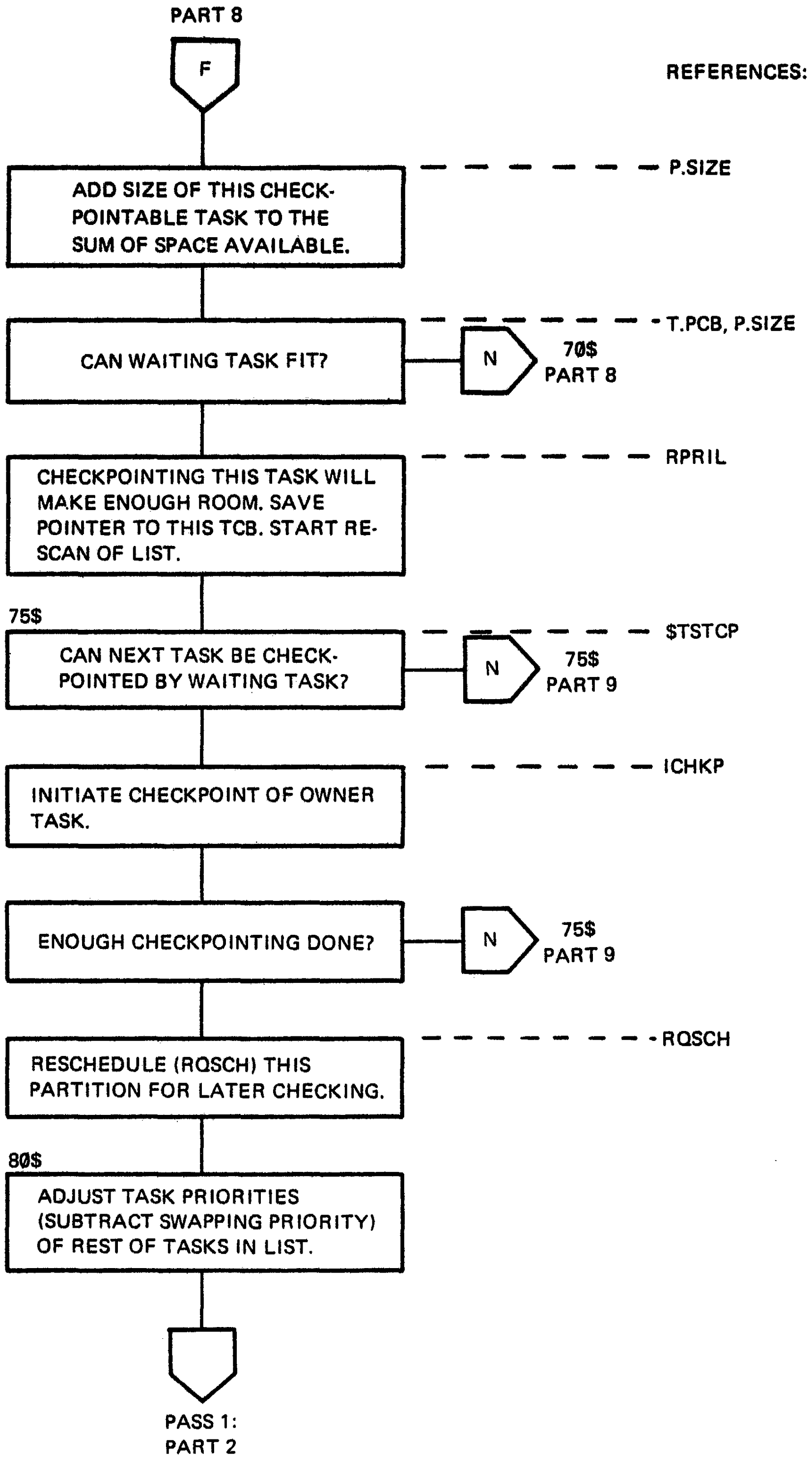

PART 8
F
REFERENCES:
P.SIZE
ADD SIZE OF THIS CHECK-POINTABLE TASK TO THE SUM OF SPACE AVAILABLE.
T.PCB, P.SIZE
CAN WAITING TASK FIT?
N
70$ PART 8
RPRIL
CHECKPOINTING THIS TASK WILL MAKE ENOUGH ROOM. SAVE POINTER TO THIS TCB. START RE-SCAN OF LIST.
75$
$TSTCP
CAN NEXT TASK BE CHECK-POINTED BY WAITING TASK?
N
75$ PART 9
ICHKP
INITIATE CHECKPOINT OF OWNER TASK.
ENOUGH CHECKPOINTING DONE?
N
75$ PART 9
RQSCH
RESCHEDULE (RQSCH) THIS PARTITION FOR LATER CHECKING.
80$
ADJUST TASK PRIORITIES (SUBTRACT SWAPPING PRIORITY) OF REST OF TASKS IN LIST.
PASS 1:
PART 2

Figure 2-12 Shuffler Logical Flow Diagram (Part 9 of 11)

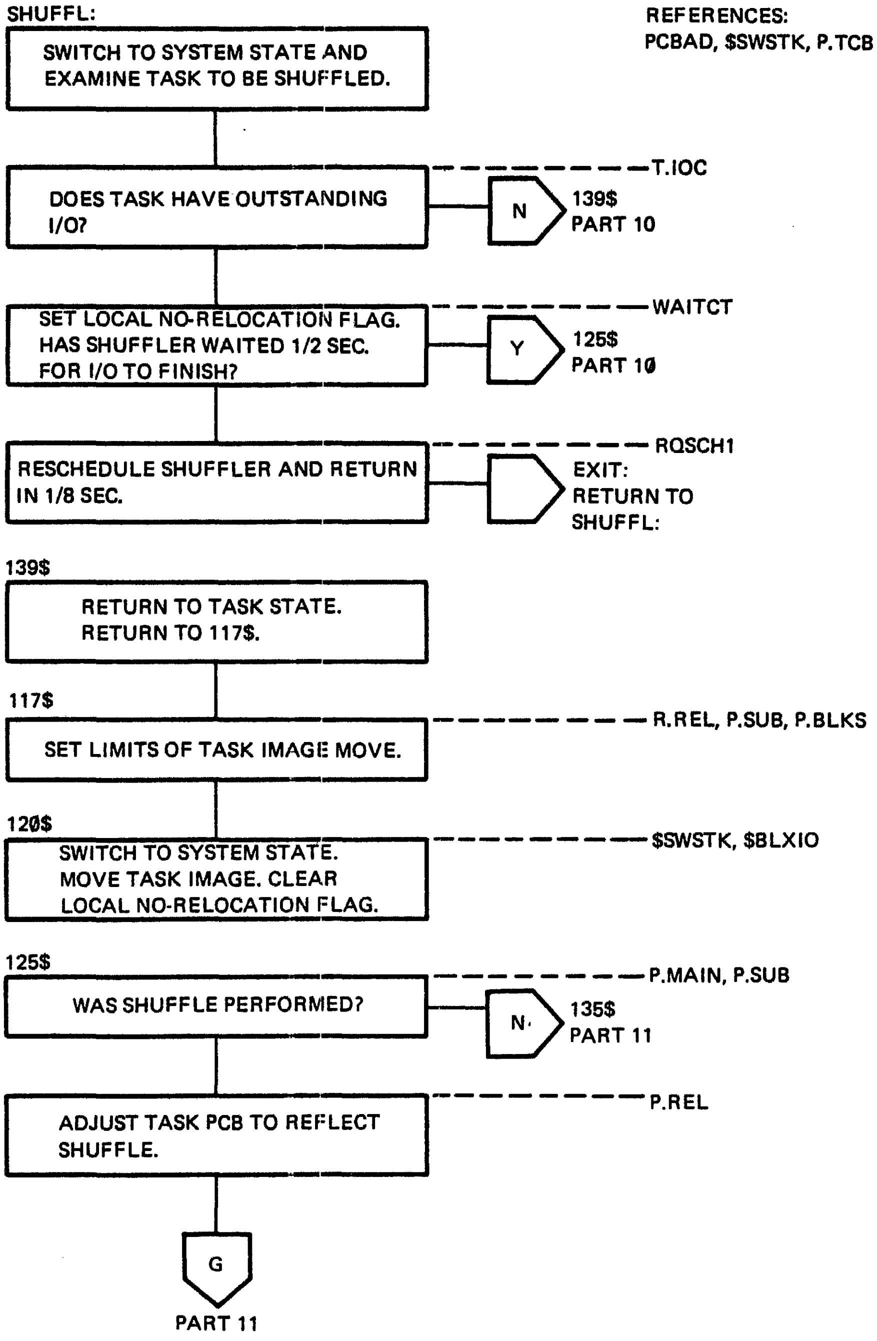

SHUFFL:
REFERENCES:
PCBAD, $SWSTK, P.TCB
SWITCH TO SYSTEM STATE AND EXAMINE TASK TO BE SHUFFLED.
T.IOC
DOES TASK HAVE OUTSTANDING I/O?
N
139$
PART 10
WAITCT
SET LOCAL NO-RELOCATION FLAG. HAS SHUFFLER WAITED 1/2 SEC. FOR I/O TO FINISH?
Y
125$
PART 10
RQSCH1
RESCHEDULE SHUFFLER AND RETURN IN 1/8 SEC.
EXIT:
RETURN TO
SHUFFL:
139$
RETURN TO TASK STATE.
RETURN TO 117$.
117$
R.REL, P.SUB, P.BLKS
SET LIMITS OF TASK IMAGE MOVE.
120$
$SWSTK, $BLXIO
SWITCH TO SYSTEM STATE. MOVE TASK IMAGE. CLEAR LOCAL NO-RELOCATION FLAG.
125$
P.MAIN, P.SUB
WAS SHUFFLE PERFORMED?
N
135$
PART 11
P.REL
ADJUST TASK PCB TO REFLECT SHUFFLE.
G
PART 11

Figure 2-12 Shuffler Logical Flow Diagram (Part 10 of 11)

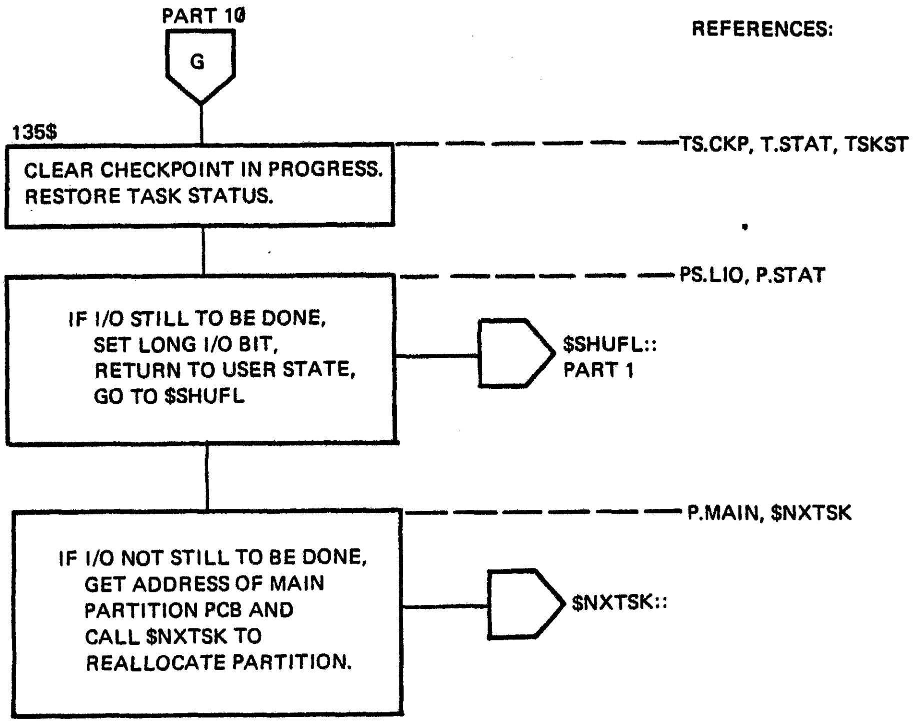


Figure 2-12 Shuffler Logical Flow Diagram (Part 11 of 11)

2.4 MEMORY ALLOCATION DATA STRUCTURES

The two fundamental data structures that are used by the Executive during memory allocation are the Partition Control Block (PCB) and the Task Control Block (TCB). They are both included here for your reference while you follow the operations of the flow diagrams.

2.4.1 Partition Control Block (PCB)

The PCB serves three major functions in the memory allocation routines:

1. The PCB contains the starting address and length of the main or subpartition it represents.

2. The PCB of a main task partition or a system-controlled partition contains busy flags and is the listhead of a linked list of subpartition PCBs. This allows the Executive to determine the availability of space within a partition.

3. The main partition PCB serves as the listhead for a linked list called the partition wait queue. Tasks that are competing for space in the partition and are out of memory have their TCBs linked into this list.

A partition or subpartition may be created in three ways:

1. By a VMR or MCR Set command
2. By the Executive in a system-controlled partition
3. By the Loader task when loading a device driver into a system-controlled partition

Whenever a partition is created, a PCB is allocated from the Dynamic Storage Region. The PCB is then filled with the starting address and length of the partition and is linked into the appropriate system lists.

Partition Control Block, Figure 2-13, describes the fields contained in the Partition Control Block.

PARTITION CONTROL BLOCK (PCB)

<table>
<tr><td colspan="2">P.LNK</td><td>LINK TO NEXT PARTITION PCB.</td></tr>
<tr><td>P.PRI</td><td>P.IOC</td><td>PRIORITY OF PARTITION; I/O AND I/O STATUS BLOCK COUNT.</td></tr>
<tr><td colspan="2">P.NAM</td><td>PARTITION NAME IN RAD5Ø.</td></tr>
<tr><td colspan="2">P.SUB</td><td>POINTER TO NEXT SUBPARTITION.</td></tr>
<tr><td colspan="2">P.MAIN</td><td>POINTER TO MAIN PARTITION.</td></tr>
<tr><td colspan="2">P.HDR</td><td>POINTER TO HEADER CONTROL BLOCK (IF M$$MGE NOT DEFINED).</td></tr>
<tr><td colspan="2">P.REL</td><td>STARTING PHYSICAL ADDRESS OF PARTITION.</td></tr>
<tr><td colspan="2">P.BLKS/P.SIZE</td><td>SIZE OF PARTITION IN BYTES.</td></tr>
<tr><td colspan="2">P.WAIT</td><td>PARTITION WAIT QUEUE LISTHEAD.</td></tr>
<tr><td colspan="2">P.SWSZ</td><td>PARTITION SWAP SIZE.</td></tr>
<tr><td colspan="2">P.BUSY (2 BYTES)</td><td>PARTITION BUSY FLAGS.</td></tr>
<tr><td colspan="2">P.OWN/P.TCB</td><td>TCB ADDRESS OF OWNER TASK.</td></tr>
<tr><td colspan="2">P.STAT</td><td>PARTITION STATUS FLAGS.</td></tr>
<tr><td colspan="2">P.HDR</td><td>POINTER TO HEADER CONTROL BLOCK (IF M$$MGE IS DEFINED).</td></tr>
<tr><td colspan="2">P.PRO</td><td>PROTECTION WORD FOR P$$LAS (DEWR, DEWR, DEWR, DEWR).</td></tr>
<tr><td colspan="2">P.ATT</td><td>ATTACHMENT DESCRIPTOR LIST HEAD (FOR P$$LAS).</td></tr>
</table>

Figure 2-13 Partition Control Block (Part 1 of 2)

Partition Status Word Bit Definitions

PS.OUT='B'100000	Partition is out of memory
PS.CKP='B'40000	Partition checkpoint in progress
PS.CKR='B'20000	Partition checkpoint is requested
PS.CHK='B'10000	Partition is not checkpointable
PS.FXD='B'4000	Partition is fixed
PS.PER='B'2000	Parity error in partition
PS.LIO='B'1000	Marked by shuffler for long I/O
PS.NSF='B'400	Partition is not shufflable
PS.COM='B'200	Library or common block
PS.PIC='B'100	Position independent library or common
PS.SYS='B'40	System controlled partition
PS.DRV='B'20	Driver is loaded in partition
PS.DEL='B'10	Partition should be deleted when not attached
PS.APR='B'7	Starting PAR number mask

Attachment Descriptor Offsets

A.PCBL:'L'.BLKW 1	PCB attachement queue thread word
A.PRI:'L'.BLKB 1	Priority of attached task
A.IOC:'L'.BLKB 1	I/O count through this descriptor
A.TCB:'L'.BLKW 1	TCB address of attached task
A.TCBL:'L'.BLKW 1	TCB attachment queue thread word
A.STAT:'L'.BLKB 1	Status byte
A.MPCT:'L'.BLKB 1	Mapping count of task through this descriptor
A.PCB:'L'.BLKW 1	PCB address of attached task
A.LGTH:'B'.	

Attachement Descriptor Status Byte Bit Definitions

AS.DEL='B'10	Task has delete access
AS.EXT='B'4	Task has extend access
AS.WRT='B'2	Task has write access
AS.RED='B'1	Task has read access

Figure 2-13 Partition Control Block (Part 2 of 2)

2.4.2 Task Control Block (TCB)

The TCB contains three major kinds of information:

1. Links and listheads to other control blocks or queues
2. Pointers related to task execution and needed by the Executive
3. Three words of status information

Other information includes:

- Task priority
- I/O pending count
- Task name
- Task local event flags
- Task default priority
- Task image size

Figure 2-14, Task Control Block, describes the fields in the Task Control Block.

<table>
<tr><td colspan="2">T.LNK</td><td>UTILITY LINK WORD.</td></tr>
<tr><td>T.PRI</td><td>T.IOC</td><td>TASK PRIORITY: I/O PENDING COUNT</td></tr>
<tr><td colspan="2">T.CPCB</td><td>POINTER TO CHECKPOINT PCB*</td></tr>
<tr><td colspan="2">T.NAM</td><td>TASK NAME IN RAD50.</td></tr>
<tr><td colspan="2">T.RCVL</td><td>RECEIVE QUEUE LISTHEAD.</td></tr>
<tr><td colspan="2">T.ASTL</td><td>AST QUEUE LISTHEAD.</td></tr>
<tr><td colspan="2">T.EFLG</td><td>TASK LOCAL EVENT FLAGS 1-32.</td></tr>
<tr><td colspan="2">T.UCB</td><td>UCB ADDRESS FOR PSEUDO DEVICE "TI"</td></tr>
<tr><td colspan="2">T.TCBL</td><td>TASK LIST THREAD WORD.</td></tr>
<tr><td colspan="2">T.STAT</td><td>FIRST STATUS WORD (BLOCKING BITS).</td></tr>
<tr><td colspan="2">T.ST2</td><td>SECOND STATUS WORD (STATE BITS).</td></tr>
<tr><td colspan="2">T.ST3</td><td>THIRD STATUS WORD (ATTRIBUTE BITS).</td></tr>
<tr><td>T.DPRI</td><td></td><td>TASK'S DEFAULT PRIORITY.</td></tr>
<tr><td colspan="2">T.LBN</td><td>LBN OF TASK LOAD IMAGE.</td></tr>
<tr><td colspan="2">T.LDV</td><td>UCB ADDRESS OF LOAD DEVICE.</td></tr>
<tr><td colspan="2">T.PCB</td><td>PCB ADDRESS OF TASK PARTITION.</td></tr>
<tr><td colspan="2">T.MXSZ</td><td>MAXIMUM SIZE OF TASK IMAGE (MAPPED SYSTEM)</td></tr>
<tr><td colspan="2">T.ACTL</td><td>ADDRESS OF NEXT TASK IN ACTIVE LIST.</td></tr>
<tr><td colspan="2">T.ATT</td><td>ATTACHMENT DESCRIPTOR LISTHEAD.</td></tr>
<tr><td colspan="2">T.OFF</td><td>OFFSET TO TASK IMAGE IN PARTITION.</td></tr>
<tr><td>RESERVED</td><td>T.SRCT</td><td>SREF WITH EFN COUNT IN ALL RECEIVE QUEUES.</td></tr>
<tr><td colspan="2">T.RRFL</td><td>RECEIVE BY REFERENCE LISTHEAD.</td></tr>
</table>

*OR LINK TO ITBS FOR TASKS CONNECTED TO INTERRUPTS.

Figure 2-14 Task Control Block (Part 1 of 2)

Task Status Definitions (* = statement true when bit is on)

First Status Word (Blocking Bits)

TS.EXE='B'100000	Task not in execution *
TS.RDN='B'40000	I/O run down in progress *
TS.MSG='B'20000	Abort message being displayed *
TS.NRP='B'10000	Task mapped to nonresident partition *
TS.RUN='B'4000	Task is running on another processor *
TS.OUT='B'400	Task is out of memory *
TS.CKP='B'200	Task is being checkpointed *
TS.CKR='B'100	Task checkpoint requested *

Task Blocking Status Mask

TS.BLK='B'TS.CKP!TS.CKR!TS.EXE!TS.MSG!TS.NRP!TS.OUT!TS.RDN

Second Status Word (State Bits)

T2.AST='B'100000	AST in progress *
T2.DST='B'40000	AST recognition disabled *
T2.CHK='B'20000	Task not checkpointable *
T2.CKD='B'10000	Checkpointing disabled *
T2.BFX='B'4000	Task being fixed in memory *
T2.FXD='B'2000	Task fixed in memory *
T2.TIO='B'1000	Task is engaged in terminal I/O *
T2.CAF='B'400	Dynamic checkpoint space allocation failure *
T2.HLT='B'200	Task is being halted *
T2.ABO='B'100	Task marked for abort *
T2.STP='B'40	Task stopped *
T2.STP='B'20	Task stopped *
T2.SPN='B'10	Saved TS.SPN on AST in progress
T2.SPN='B'4	Task suspended *
T2.WFR='B'2	Saved TS.WFR on AST in progress
T2.WFR='B'1	Task in waitfor state *

Third Status Word (Attribute Bits)

T3.ACP='B'100000	Ancillary control processor *
T3.PMD='B'40000	Dump task on synchronous abort
T3.REM='B'20000	Remove task on exit *
T3.RPV='B'10000	Task is privileged *
T3.MCR='B'4000	Task was requested as external MCR function *
T3.SLV='B'2000	Task is a slave task *
T3.CLI='B'1000	Task is command line interpreter *
T3.RST='B'400	Task is restricted *
T3.NSD='B'200	Task does not allow send data
T3.CAL='B'100	Task has checkpoint space in task image
T3.ROV='B'40	Task has resident overlays
T3.NET='B'20	Network protocol level

Figure 2-14 Task Control Blocks (Part 2 of 2)

CHAPTER 3

INTERRUPT PROCESSING

3.1 INTRODUCTION

This chapter discusses the internal operation of the RSX-11M interrupt mechanisms. Flow diagrams of important routines are included in this chapter.

3.2 INTERRUPT MECHANISMS

RSX-11M is a priority driven, multiprogramming, real-time operating system. As with any such system, its principle function is multiplexing the sharable resources among competing tasks. The multiplexing itself is made possible by the interrupt system of the hardware that causes control to be taken away from user tasks and given to the Executive. It is during this period of interrupt control that the Executive makes decisions about granting use of shared resources. Understanding the interrupt mechanism is fundamental to understanding the Executive. Once this is understood, the knowledge serves as a framework for describing the operation of Executive subsystems (drivers, I/O, etc.) and the system as a whole.

3.2.1 Hardware Interrupt Mechanisms - Review and Overview

The PDP-11 family of computers has two classes of interrupts:

1. Processor traps

2. External interrupts

Processor traps cannot be masked (blocked) in any way by altering the priority of the processor. When processor traps occur the processor enters the trap sequence of pushing the PS and PC onto the current stack (system or user) and retrieving the PS and PC from the proper hardware trap vector. If no other interrupts are pending when this occurs, the processor then begins at the location specified by the trap vector. A table of trap vectors starts at location 0 in low memory and extends to location 774(8). However, RSX-11M does not use locations 0 and 2 as vectors. Processor traps include the:

Breakpoint trap (BPT) instruction

Emulator trap (EMT) instruction

Input/Output Trap (IOT) instruction

TRAP instruction

11/40 floating-point exception fault

Odd address

Power fail

Illegal instruction

External interrupts are hardwired to one of the four bus request levels of the processor. These interrupts are generally associated with I/O devices and are maskable. They can only cause an interrupt when the priority in the Processor Status Word (PS) is less than the priority of the interrupting source. Thus, by setting the processor priority in a trap vector PS word to an appropriate level, interrupts equal to or below that priority are locked out.

Every device that causes an interrupt has an associated trap vector in the vector table located in lower memory. However, not all devices cause interrupts, therefore, those devices do not have associated trap vectors.

3.2.2 Executive and Stack Processing

All the vectors in the trap vector table must be initialized properly so that when a processor trap or interrupt occurs, an Executive interrupt routine obtains control of the processor.

On an unmapped PDP-11, only one stack exists. This stack must be multiplexed to service the user tasks and the Executive. Having only a single stack also implies a single processor state. The Executive must simulate a two state system. A single word, the stack depth indicator ($STKDP) is used to control this simulation.

On a mapped system, there are two stacks - the user stack and the system stack.

When the word, $STKDP, is equal to 1 the system is running in the user state; when it is zero or less, it is in the system state. All stack multiplexing is accomplished by testing the contents of this word. Note that the priority set in the PS word for user tasks (both privileged and unprivileged) is 0, and for Executive routines, when running interruptable, is either 0, 7 or the level at which the interrupt was taken. These priorities play an important part in the goal to operate the Executive and its associated routines non-interruptable for as short a duration as possible.

Describing the RSX-11M interrupt mechanism involves several interrelated routines, and it may be necessary for you to read the following section twice before the process becomes completely clear.

3.3 INTERRUPT PROCESSES

The RSX-11M interrupt machinery involves the following routines or routine types:

Interrupt processor (both external interrupts and traps);

The Interrupt Save Routine ($INTSV);

The Directive Save Routine ($DIRSV);

The Interrupt Exit Routine ($INTXT);

The Directive Exit Routine ($DIRXT); and

The Fork Processors ($FORK,$FORK0,$FORK1).

For resident drivers only, the device interrupt vector must be initialized when defining data structures, and not dynamically. This practice makes the driver code independent of device register address assignments and of the actual location of the interrupt vector. The driver data structure must include a storage assignment and initialization for the interrupt vector with the priority set to PR7.

Writers of loadable drivers do not initialize the device interrupt vector. The vector is dynamically established by Load when the driver is loaded. When a driver is unloaded, Unload sets the vector to the system nonsense interrupt entry point.

Driver interrupt processing routines are entered directly from the vector and usually use the INTSV$ macro for state switching services; at the completion of these services, the interrupt routines are again given control to complete the interrupt service. The exit routines $INTXT and $DIRXT restore the state prior to switching to the system state, control the unnesting of interrupts, and make checks on the state of the system (for example, is it necessary to redispatch the processor). The Fork processing routines linearize access to shared system data bases. The details of all these routines are discussed later in the text.

3.3.1 The INTSV$ Macro

INTSV$ is a system macro that minimizes coding differences between loadable and resident drivers. INTSV$ contains conditionally assembled code to handle:

1. Single or multiple controllers

2. Loadable or resident drivers

3. Mapped or unmapped systems

This macro is required for loadable drivers on mapped systems, because interrupts from hardware devices must be processed in kernel address space. In particular, the decoding of the PS word and the call to $INTSV must be done before entering the driver. Thus, a call to the Executive routine $INTSV within a loadable driver is illegal, and the MCR Load function returns an error if loading is attempted.

When the INTSV$ macro is used for a loadable driver in a mapped system, the Load function allocates a block of dynamic memory in kernel address space to contain the interrupt coding. This block, called the Interrupt Control Block (ICB), also contains coding to:

1. Save the kernel mapping (APR5)

2. Load APR5 to map the driver

3. Transfer to the driver

4. Restore the mapping after return

The Load function also sets up the controller's interrupt vector so that hardware interrupts point to the ICB.

Finally, the use of the INTSV$ macro in a loadable driver on a mapped system requires that the symbol LD$xx (where xx is the 2-character device mnemonic) be defined either in the driver source or the assembly prefix file RSXMC.MAC.

3.3.1.1 **INTSV$ Macro Format** - The format of the INTSV$ macro is:

```
     INTSV$ xx,pri,nctlr[,pssave,ucbsave]
```

where:

xx — is the 2-character device mnemonic.

pri — is the priority of the device (the priority that would be used in a call to $INTSV).

nctlr — is the number of controllers the driver services.

pssave — is an optional argument specifying a variable in which to save the PS word. If omitted, a variable named TEMP is used.

ucbsave — is an optional argument specifying a block of contiguous words in which to retrieve the interrupting device's UCB address. If omitted, a block of contiguous words named CNTBL is used.

Outputs: R4 — is the controller index, only if nctlr is greater than 1.

R5 — is the UCB address.

Example:

```
          INTSV$ PP,PR4,P$$P11
```

3.3.2 **External Interrupt from the Task State ($STKDP=1)**

The vectors in lower memory contain a PC unique to each interrupting source, and a PS set with a priority of PR7. Hence, when an external interrupt occurs, the hardware pushes the current PS and PC onto the current stack (in this case the task's stack) and loads the new PC and PS (set at PR7) from the appropriate interrupt vector. The interrupt routine then starts executing with interrupts locked out. Interrupt routines may, in fact, be executing in one of three states:

1. At PR7 with interrupts locked out;

2. At the priority of the interrupting source; thus, interrupt levels greater than the source are permitted, or

3. At Fork level which is at PR0.

By internal convention, processing in the PR7 state is limited to 100us. If processing can be completed in this time, then the interrupt routine simply RTI's; the interrupt has been processed and dismissed with minimal overhead.

If the interrupt routine requires additional processing time (but does not intend to alter a shared system data base) it uses the INTSV$ macro. The priority at which the caller is to run is included in the INTSV$ macro or the call to $INTSV. With loadable drivers the Interrupt Control Block calls $INTSV. Therefore, the driver cannot use a CALL to $INTSV; it must use the INTSV$ macro.

The interrupt save routine, $INTSV, uses the priority specified in the INTSV$ macro line (the interrupting source priority) to set up the interrupt routine. At this point in the process, the interrupt routine is interruptable by devices with priorities higher than that of the interrupting source. The $INTSV routine then conditionally switches to system state if the processor is not already in system state.

3.3.2.1 **$INTSV Routine** - The $INTSV algorithm is:

```
$INTSV:  Push R5 and R4 onto the current stack.

         Decrement stack depth indicator, $STKDP.

         Is the stack depth indicator =0?  No;  go to 1.

         Save the current (a task's in this case) stack pointer.

         Set up the System stack pointer (switch stacks if not
         M$$MGE).

     1.  Load the new processor priority as specified by the caller.

         Return to caller.
```

Note:

The stack depth indicator, $STKDP, is zero only after the transition from the user state to the system state occurs.

The JSR R5,$INTSV instruction pushes R5 on the stack prior to entering the $INTSV routine. Pushing of R4 and R5 is done to free these registers for routines processing external interrupts. It is an internal programming convention that assumes these routines will not require more than two registers to accomplish their functions. If they do, they must save and restore any additional registers they use.

3.3.2.2 **INTSV$ Macro** - The interrupt save macro, INTSV$, expands as shown in Figure 3-1.

3.3.3 External Interrupts from the System State ($STKDP <=0)

The code on this interrupt path is identical to that discussed in External Interrupt from the Task State. However, it is not necessary for the task to switch states when the INTSV$ macro is used. The current stack is the system stack, and when $INTSV tests the value of the stack depth indicator, $INTSV bypasses saving the SP and switching the stacks. After $INTSV saves R4 and R5 on the system stack, it returns to the driver interrupt routine.

```
.MACRO INTSV$ DEV,PRI,NCTRLR,PSWSV,UCBSV
.IF NDF L$$DRV ! M$$MGE ! LD$'DEV
.IF GT NCTRLR-1
.IF B <PSWSV>
MFPS TEMP
.IFF
MFPS PSWSV
.ENDC
.IFTF
JSR R5,$INTSV
.IF DF L$$SI1
.WORD PRI
.IFF
.WORD ^C<PRI>&PR7
.ENDC
.IFT
.IF B <PSWSV>
MOV TEMP,R4
.IFF
MOV PSWSV,R4
.ENDC
BIC #177760,R4
ASL R4
.ENDC
.ENDC
GTUCB$ UCBSV,NCTRLR
.ENDM
```

Figure 3-1 INTSV$ Macro Expansion

```
;+
; **-$PPINT-PC11 PAPER TAPE PUNCH CONTROLLER INTERRUPT ROUTINE
;-

$PPINT::                                ;;;REF LABEL
        INTSV$  PP,PR4,P$$P11           ;;;GENERATE INTERRUPT SAVE CODE
        MOV     U.SCB(R5),R4            ;;;GET ADDRESS OF SCB
        MOVB    S.ITM(R4),S.CTM(R4)     ;;;RESET TIMEOUT COUNT
        MOV     S.CSR(R4),R4            ;;;POINT R4 TO CSR
        MOV     (R4)+,U.CW3(R5)         ;;;SAVE STATUS
        BMI     60$                     ;;;IF MI, ERROR
        SUB     #1,U.CNT(R5)            ;;;DECREMENT CHARACTER COUNT
        BCS     50$                     ;;;IF CS, THEN DONE
        TSTB    U.CW2(R5)               ;;;CURRENTLY PUNCHING TRAILER?
        BPL     30$                     ;;;IF PL NO
        CLRB    (R4)                    ;;;LOAD NULL INTO OUTPUT REGISTER
        BR      40$                     ;;;BRANCH TO LOAD IT
30$:    CALL    $GTBYT                  ;;;GET NEXT BYTE FROM USER BUFFER
        MOVB    (SP)+,(R4)              ;;;LOAD BYTE INTO OUTPUT REGISTER
40$:    JMP     $INTXT                  ;;;EXIT FROM INTERRUPT
50$     INC     U.CNT(R5)               ;;;RESET BYTE COUNT
60$:    CLR     -(R4)                   ;;;DISABLE PUNCH INTERRUPTS
        CALL    $FORK                   ;;;CREATE SYSTEM PROCESS
        MOV     U.SCB(R5),R4            ;POINT R4 TO SCB
        MOV     S.PKT(R4),R1            ;POINT R1 TO I/O PACKET
        MOV     I.PRM+4(R1),R1          ;AND PICK UP CHARACTER COUNT
        SUB     U.CNT(R5),R1            ;CALCULATE CHARACTERS TRANSFERRED
        MOV     #IS.SUC&377,R0          ;ASSUME SUCCESSFUL TRANSFER
        TST     U.CW3(R5)               ;DEVICE ERROR?
        BPL     70$                     ;IF PL NO
65$:    MOV     #IE.VER&377,R0          ;UNRECOVERABLE HARDWARE ERROR CODE
70$:    CALL    $IODON                  ;INITIATE I/O COMPLETION
        BR      PPINI                   ;BRANCH BACK FOR NEXT REQUEST
```

Figure 3-2 Example of a Driver Using $INTSV

3.3.4 Processor Traps from the Task State ($STKDP<=1)

when a processor trap occurs from the task state, the hardware pushes PS, PC, and initiates the routine specified in the associated hardware trap vector. If an Executive directive causes the trap, EMT 377, the Directive Parameter Block (DPB) or its address was pushed onto the user task's stack prior to the issuance of the EMT.

Also, the task can cause a processor trap by issuing the SWSTK$ macro. See Chapter 4 for an explanation of the SWSTK$ macro.

The trap routine, running at PR7, immediately calls the routine $DIRSV (Directive Save), which has the following algorithm:

```
$DIRSV:  Push R5 and R4 onto current stack

         Decrement stack depth indicator.

         Is the stack depth indicator <=0?  No, go to 1.

         Save current task's stack pointer.

         Set up system stack pointer (switch stacks if not M$$MGE).

    1.   Push R3, R2, R1, R0 onto current (system) stack.

         Load new processor priority as specified by the caller.

         Return to caller.
```

The $STKDP check is made to improve crash analysis; no other decisions are made in $DIRSV because all processor traps, with the two exceptions of the Trap instruction or Powerfail, occur from the task state. The exceptions are handled on exit. All registers are saved; the need for only two registers, R5 and R4 is assumed only for routines processing external interrupts. As with $INTSV, the priority at which the caller expects to run immediately follows the call. All processor trap routines, however, run interruptable.

Only one processor trap can be queued for processing in the system at any point in time (ignore, for the moment, the two exceptions we have noted). Because the processor trap occurred in task state, entrance to the Executive occurs only when the Executive is idle. While in the system state, only external interrupts can occur. If processor traps occur, then either they are valid exceptions, or the system itself has faulted and shuts down.

Once a valid processor trap is pending, it is processed to completion before any other system routine is given access to any shared system data base. This strict sequentiality is accomplished with the two exit routines $INTXT, $DIRXT and the fork processors ($FORK, $FORK0, and $FORK1).

3.3.4.1 **Example use of $DIRSV** - Figure 3-3 shows the code for the Emulator Trap (EMT) processing routine, $EMTRP.

```
;+
;  EMT TRAP PROCESSING ROUTINE
;
;  THIS ROUTINE IS ENTERED VIA THE VECTOR AT LOCATION 30 WHEN AN EMT
;  INSTRUCTION IS EXECUTED.  THE ROUTINE IS ENTERED IN SYSTEM STATE.
;  IF THE STACK DEPTH IS NOT 0 AFTER THE DIRSV$ MACRO EXECUTES,
;  THE SYSTEM CRASHES.
;-

$EMTRP::DIRSV$                      ;;;SAVE REGISTERS AND SET PRIORITY
             .MACRO DIRSV$
             JSR R5,$DIRSV
             .ENDM
        TST        $STKDP           ;WERE WE AT STACK DEPTH +1
        BNE        70$              ;IF NE 0 - NO - CRASH SYSTEM
        MOV        @$HEADR          ;GET SAVED STACK POINTER
        CMP        (R3)+,(R3)+      ;POINT TO USER PC WORD
        MOV        (R3)+,R5         ;GET ADDRESS OF EMT +2

        .IF DF M$$MGE

        MFPI       -(R5)            ;GET DIRECTIVE WORD
        CMP        #104377,(SP)     ;DIRECTIVE EMT 377?
        BNE        80$              ;IF NE 0 -NO-
        MOV        #1,(SP)          ;SET SUCCESSFUL DIRECTIVE STATUS

        .IFF

        CMP        #104377,-(R5)    ;DIRECTIVE EMT 377
        BNE        80$              ;IF NE 0 -NO-
        MOV        #1,-(SP)         ;SET SUCCESSFUL DIRECTIVE STATUS

        .IFTF

        MOV        #USRPS,R5        ;POINT TO LOCAL DATA
        MOV        R3,(R5)+         ;SAVE ADDRESS OF USER PS
        BIC        (SP),(R3)+       ;CLEAR CARRY IN USER PS WORD
        CLR        (R5)             ;INDICATE NO BYTES
        .
        .
        .
80$:
        .IF        DF M$$MGE

        MOV        $HEADR,R5        ;POINT TO CURRENT TASK HEADER
        MOV        H.WND(R5),R5     ;POINT TO NUMBER OF WINDOW BLOCKS
        TST        W.BLVR+2(R5)     ;CURRENT TASK MAPPED TO EXEC
        BEQ        85$              ;IF EQ 0 -NO-
        CMP        (SP),#104376     ;IS THIS A CALL TO $SWSTK

        .IFF

        CMP        (R5),#104376     ;IS THIS A CALL TO $SWSTK

        .ENDC

        BNE        85$              ;IF NE 0 -NO-
        JMP        $SWSTK           ;PROCESS CALL TO $SWSTK
85$:    JMP        $EMSST           ;PROCESS SST FAULT
                                    ;
```

Figure 3-3 Example of Use of $DIRSV by the $EMTRP Routine

```
$DIRSV::MOV     R4,-(SP)        ;;;SAVE R4
        DEC     $STKDP          ;;;SET PROPER STACK DEPTH
        BNE     10$             ;;;IF NE, DON'T SWITCH STACKS
        MOV     SP,@$HEADR      ;;;SAVE CURRENT SP

        .IF NDF M$$MGE

        MOV     #$STACK,SP      ;;;LOAD SYSTEM STACK POINTER

        .ENDC

10$:    MTPS    #0              ;;;ALLOW INTERRUPTS
        MOV     R3,-(SP)        ;SAVE REGISTERS R3-R0 ON STACK
        MOV     R2,-(SP)        ;
        MOV     R1,-(SP)        ;
        MOV     R0,-(SP)        ;
        CALL    (R5)            ;CALL SYNCHRONOUS TRAP ROUTINE
        BR      $DIRXT          ;EXIT FROM TRAP
```

Figure 3-3 (Cont.) Example of Use of $DIRSV by the $EMTRP Routine

3.3.5 Processor Traps from the System State ($STKDP <=0)

Only two processor traps are valid from the system state: the Trap instruction and Powerfail. If any other processor trap occurs while in the system state, the system's operation is aborted or XDT, the Executive debug tool, is entered if it is present.

3.3.5.1 **Processing for Trap Instructions Occurring in System State** - The Executive uses the trap instruction as a core saving technique in returning status following the execution of an Executive directive. The EMT 377, which is the processor trap used to initiate directives, causes entry into the directive dispatcher ($EMTRP) which in turn calls $DIRSV. See Figure 3-3. On return from $DIRSV, but before calling the directive processing routine, the directive dispatcher pushes a value of +1 onto the system stack, and clears the C bit in the PS word stored on the user's stack. It then calls the proper directive processing routine to execute the directive. Figure 3-4, Stack Stack Upon Entry into Directive Processing, shows the state of the user and system stacks for both the unmapped and mapped systems at the time entry is made to the routine that processes the issued directive.

The directive processing routine now carries out its function, and in so doing is free to alter any shared system data base, because no other routine can gain access to a shared data base until the directive processing routine is completed. This arrangement of the stack and interface between the directive dispatcher and the directive processors has two advantages:

1. The normal return for all but a few directives is a +1 status and carry clear. This means the directive routines can return to the dispatcher with an RTS; thus the return path is one word rather than the two needed if a JMP is employed; this scheme probably saves 100 words in the RSX-11M Executive.

2. Internal Executive routines can call the directive processing routines without using an EMT.

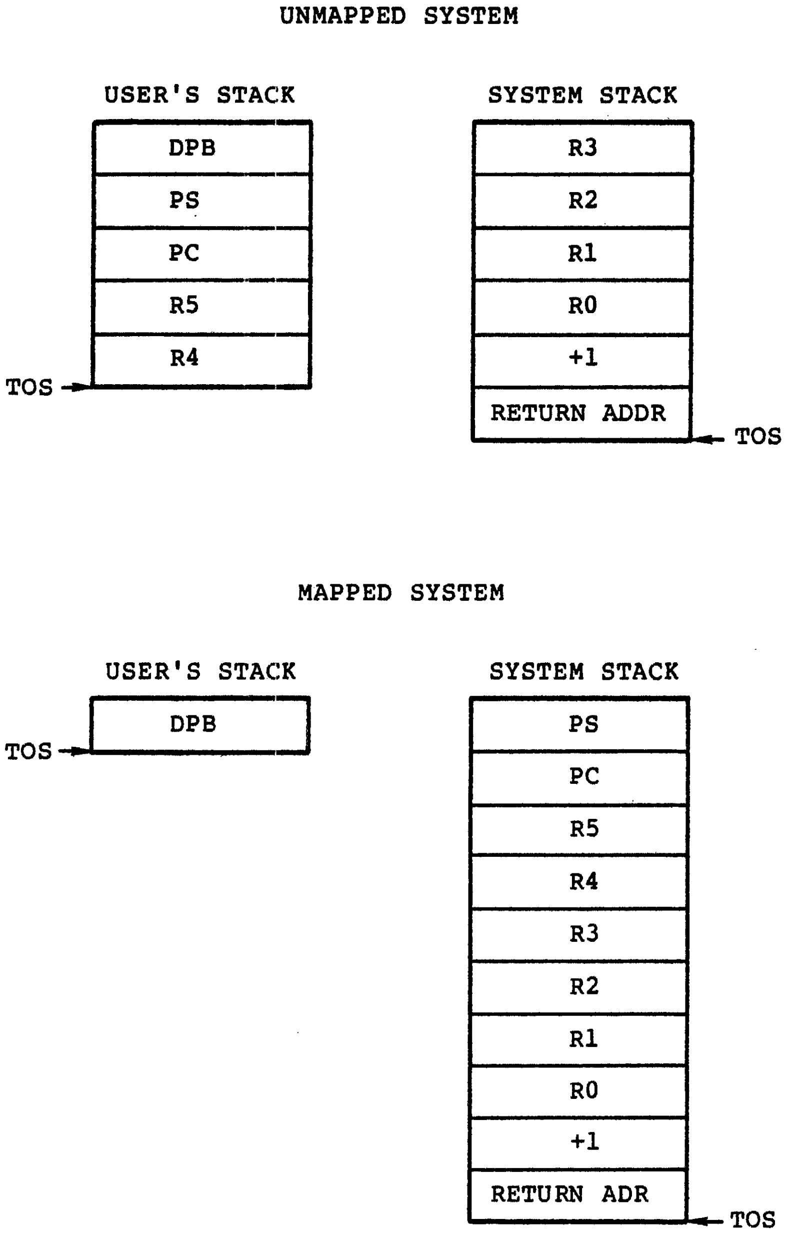


Figure 3-4 Stack State Upon Entry into Directive Processing

If a directive processing routine needs to return a status other than +1, and have the carry stat clear, the routine replaces the +1 on the stack with the value it intends to return and then executes an RTS.

Now to the use of the Trap instruction within the Executive. If a directive processing routine must return a status other than +1 and, in addition have the carry stat set, or cleared, based on the status value returned, it then uses the Trap instruction with the value of the status to be returned in the low order byte of the instruction. When the trap processing routine is entered, it immediately checks for stack depth=0, and if 0, proceeds to reset the stack for correct exiting from a directive processing routine. The low order byte of the trap instruction itself overlays the +1 status currently on the stack; this value is tested and, if minus, the carry stat is set in the user task's PS word. If plus, the carry stat is left cleared. After this processing, the exiting code of the directive dispatcher is entered just as if the directive processing routine had executed an RTS.

If the initial test for a stack depth indicator of 0 fails, the trap processing routine calls $DIRSV. This call is logically incorrect if the stack depth indicator was less than zero. This programming error is recognized on exit. On return from $DIRSV, the trap processing routine checks the stack depth indicator, and if it is not zero, the system crashes.

Note that directives are legitimate only from the task state (stack depth indicator=1) so that during directive processing, the stack depth indicator is always 0. Interrupts that occur in system state disappear from the stack before the directive processing sequence resumes following an interrupt. Hence, even though the stack can grow while a directive processing routine is in control, this growth is transparent to the routines. Stating it from a different perspective, interrupts are permitted but the directive processing routine that is in control is unaware of them.

Thus, directive processing routines have three methods of returning status:

1. For the normal return carry clear and status equal to +1, they use an RTS.

2. For carry clear and status other than +1, they overlay the +1 status on the stack with the desired status value (status value is at 2(SP)) and RTS.

3. For carry clear or set, and status of one byte, they use the trap instruction. This requires more overhead than 1 and 2 above but saves memory, and, of course is the required return mechanism if carry is to be set.

Together, these return mechanisms from directive processing routines save between 200 and 300 words in the RSX-11M Executive as compared to returning via jump instructions.

3.3.6 Powerfail Processing

When a power failure occurs, the power failure trap processing routine, PDOWN: - in the POWER module, is entered. This routine saves the state of the system, sets up a new power failure trap-vector PC for use when power is restored, calls the user's powerfail routine if it's defined, then halts.

On restoration of power, the state of the system at the time the failure occurred is restored, the $PWRFL flag is set indicating that a power failure has occurred, the reschedule pointer $RQSCH is set, and the clock is re-enabled. Then, the restoration code issues an RTI, which results in the resumption of the processing that was in progress when the power failure occurred. The specific processing to reflect the occurrence of a power failure does not occur until either Directive Exit is entered or the clock interrupts. In any event, this processing is part of Directive Exit and is discussed under Directive Exit.

Note that power failure processing is not asynchronous. As much as 1/60 of a second could elapse following restoration before the power failure is acted upon. The records and logic needed to provide asynchronous power failure processing are simply too large for a system with the memory objectives of RSX-11M.

3.3.7 Processing Within Interrupt Routines

In this section, we detail the events that take place following interrupt entry up to the point where the Executive is ready to return control to the task state.

Once the Executive is entered via an interrupt (regardless of the state it is in when the interrupt occurs) it does not again return to the task state until all system related processing for that interrupt has been completed.

A single interrupt in the task state causes transfer into the system state where the system remains until the interrupt is processed. But while in the system state, repeated interrupts can occur. This implies a fixed interrupt depth of one for the task stack (requiring a task to provide a stack of at least four words in an unmapped system), and implies a variable interrupt depth for the system stack.

Because multiple interrupts can occur in the system state, RSX-11M resolves both of these logical difficulties by strictly linearizing interrupt processing and access to internal data bases. The mechanisms employed to accomplish this linearization are the system stack, fork processes, and the associated fork list.

3.3.7.1 **Queuing Interrupts on the System Stack** - In the system state, the system must operate interruptable as much of the time as possible. Three possible conditions can exist when the system itself runs non-interruptable:

1. The most recent interrupt is being processed at level PR7 and the driver interrupt routine has not yet returned to an interruptable state.

2. The interrupt routine has dropped from level PR7 to the level at which the interrupt occurred. Priority levels, equal to or less than the priority of the interrupting source are locked out.

3. The system is updating a critical list whose consistency can only be maintained by a non-interruptable instruction sequence. There are two such lists, and we will discuss them shortly.

In the sections External Interrupts from the Task State and Processor Traps from the Task State, we examined the code sequence for processing external interrupts and processor traps, as well as the stack additions that occurred during their processing. Interrupt stacking in the system state occurs based principally on hardware interrupt levels. Thus, if a level PR4 interrupt is being processed, a level PR5, PR6, or PR7 interrupt can potentially interrupt this processing and cause context to be stacked and control given to the higher level interrupt routine.

3.3.8 Fork Processing

Once a driver interrupt routine passes from a non-interruptable to an interruptable state by using a call to $INTSV or the INTSV$ macro, processing is at the same level as the priority of the interrupting source. However, along any given interrupt path, more processing is often required than the minimum non-interruptable code sequences in the Executive permit. Along this path the allowable maximum non-interruptable processing time is 500us. Thus, a scheme is required to split interrupt processing routines further, such that part of their execution runs interruptable to any interrupting source. The mechanism for achieving this split is called fork processing.

First, and most important, fork processing linearizes access to system data bases. Thus eliminating unwanted recursion and untimely updates of these data bases. A list associated with fork processing, the Fork List, is the method the system uses to linearize data base accessing.

Driver and system interrupt routines are required to adhere to the following internal conventions:

1. Use of any registers except R4 and R5 requires that these registers be saved and restored.
2. Non-interruptable processing must not exceed twenty instructions.
3. All modifications to system data bases must be done via a fork process.

Along an interrupt path, control can be taken from a routine only due to a higher priority interrupt pending in the hardware. As discussed previously, these interrupts are kept track of on the system stack. When an interrupt routine needs to transfer from a non-interruptable to an interruptable state, or modify a system data base, it must call $FORK. $FORK, however, cannot be called directly from an interrupt routine; it must first switch to system state by calling $INTSV and then calling $FORK.

By virtue of calling $FORK, the routine is now interruptable and its access to system data bases is strictly linear. The Fork List is a list of system routines, usually I/O drivers, waiting to complete their processing, in particular, waiting to access a shared system data base.

When the $FORK routine returns to $INTX1 after placing the fork block in the fork list, $INTX1 and $DIRXT remove the stacked items for the driver interrupt routine. In effect the fork list is a secondary interrupt queue (stack) whose members are processed FIFO, and obtain processing time only if the system stack is empty.

Note that the context saved for a caller of $FORK depends on which entry point is called ($FORK or $FORK1), and the context saved is all that is needed to restart routines on the fork list.

Figure 3-5 Example Driver Interrupt Routine shows the expansion of the INTSV$ macro and the call to $FORK.

```
;+
; $XXINT DISK CONTROLLER INTERRUPT ROUTINE
;-
        INTSV$  DK,PR5,R$$K11   ;;;SAVE REGISTERS AND SET PRIORITY

               .MACRO INTSV$ DEV,PRI,NCTRLR,PSWSV,UCBSV
               .IF NDF L$$DRV ! M$$MGE ! LD$'DEV
               .IF GT NCTRLR-1
               .IF B <PSWSV>
               MFPS TEMP
               .IFF
               MFPS PSWSV
               .ENDC
               .IFTF
               JSR R5,$INTSV
               .IF DF L$$SI1
               .WORD PRI
               .IFF
               .WORD ^C<PRI>&PR7
               .ENDC
               .IFT
               .IF B <PSWSV>
               MOV TEMP,R4
               .IFF
               MOV PSWSV,R4
               .ENDC
               BIC #177760,R4
               ASL R4
               .ENDC
               .ENDC
               GTUCB$ UCBSV,NCTRLR
               .ENDM
        TSTB    RTTBL+1(R4)     ;;;DRIVE RESET IN PROGRESS?
        BEQ     50$             ;;;IF EQ NO
        MOV     R4,-(SP)        ;;;SAVE CONTROLLER INDEX
        MOV     U.SCB(R5),R4    ;;;GET ADDRESS OF SCB
        MOV     @S.CSR(R4),R4   ;;;GET CONTENTS OF CSR
        BMI     40$             ;;;IF MI DRIVE RESET ERROR
        BIT     #20000,R4       ;;;DRIVE RESET COMPLETE?
        BNE     40$             ;;;IF NE YES
        TST     (SP)+           ;;;CLEAN STACK
        RETURN                  ;;;
40$:    MOV     (SP)+,R4        ;;;RESTORE CONTROLLER INDEX
50$:    CALL    $FORK           ;;;CREATE A SYSTEM PROCESS
;+
;  CONTROL IS REGAINED AT THIS POINT WITH ALL INTERRUPTS ALLOWED
;-
        MOV     R4,R3           ;COPY CONTROLLER INDEX
        MOV     U.SCB(R5),R4    ;GET ADDRESS OF SCB
        MOV     S.CSR(R4),R2    ;GET ADDRESS OF CSR
        MOV     #IS.SUC&377,R0  ;ASSUME SUCCESSFUL TRANSFER
        MOV     S.PKT(R4),R1    ;GET I/O PACKET ADDRESS
        BITB    #IQ.UMD,I.FCN(R1) ;DIAGNOSTIC FUNCTION EXECUTED?
51$:    BNE     130$            ;IF NE YES
        .
        .
```

Figure 3-5 Example Driver Interrupt Routine

3.3.8.1 **$FORK** - $FORK is in the file SYSXT. A driver calls $FORK to switch from a partially interruptable level (its state following a call on $INTSV) to a fully interruptable level.

Notes:

1. $FORK cannot be called unless $INTSV has been previously called. The fork-processing routine assumes that $INTSV has set up entry conditions.

2. A driver's current timeout count is cleared in calls to $FORK. This protects the driver from synchronization problems that can occur when an I/O request and the timeout for that request happen at the same time. After a return from a call to $FORK, a driver's timeout code will not be entered.

   If the clearing of the timeout count is not desired, a driver has two alternatives:

   a. Perform timeout operations by directly inserting elements in the clock queue (refer to the description of the $CLINS routine).

   b. Perform necessary initialization, including clearing S.STS in the SCB to zero (establishing the controller as not busy), and call the $FORK1 routine rather than $FORK. Calling $FORK1 bypasses the clearing of the current timeout count.

3. The driver must not have any information on the stack when $FORK is called.

3.3.8.2 **$FORK1** - $FORK1 is the file SYSXT. A driver calls $FORK1 to bypass the clearing of its timeout count when it switches from a partially interruptable level to a fully interruptable level (refer also to the description of the $FORK routine).

Notes:

1. For mapped systems with loadable driver support, a 5-word fork block is required for calls to $FORK1.

2. When a 5-word fork block is used, the driver must initialize the fifth word with the base address (in 32-word blocks) of the driver partition. This address can be obtained from the fifth word of the standard fork block in the SCB.

3. The driver must not have any information on the stack when $FORK1 is called.

3.3.9 Exiting the System State

Two routines $INTXT (Interrupt Exit) and $DIRXT (Directive Exit) result in the sequential removal of all items on the system stack, followed by all items on the Fork List. The following text discusses these two routines.

The Executive's objective is to return to the idle state as fast and as efficiently as possible. It does this by servicing all routines on the system stack first. These routines are usually running at some level of non-interruptability. When the system stack is cleared of pending requests, the Executive then services the pending requests on the Fork List. When both the Fork List and system stack are empty, the Executive either returns to the task state or if no task is active, waits for work to do (idles).

$INTXT is transferred to by external interrupt processing routines that are running on the system stack at the priority of the interrupting source.

$DIRXT has the task of servicing the Fork List and, when the Executive has no more work to do, restoring the task state. $DIRXT is entered by trap routines, fork routines, and by $INTXT.

3.3.9.1 **$INTXT Routine** - The $INTXT algorithm is as follows:

```
$INTXT::  Lock out interrupts.

          Is $STKDP=0?  No, go to 1.

          Is Fork List empty (check $FRKHD)?  No, reload user SP if
          memory management is not defined and go to 1.

          Allow interrupts.

          Store R3, R2, R1, R0 on the current (system in this case)
          stack.

          Execute $DIRXT (Directive Exit).

      1.  Increment stack depth indicator.

          Restore R4 and R5 from current stack and RTI.
```

Notes:

Interrupts must be locked out to insure a consistent check of $STKDP and the contents of the Fork List. The same type of lockout occurs in directive exit. There are two non-interruptable code spans used to check and update the Fork List. One is in $FORK, and one in $DIRXT. The saving of R3 thru R0 is preparatory to the jump to $DIRXT, which expects these registers on the stack. Note that the path through the Executive that finds both the Fork List empty and the stack depth indicator equal to 0 is fairly common. This is the minimum overhead path.

3.3.9.2 **Directive Exit** - The $DIRXT algorithm is as follows:

```
$DIRXT::  Lock out interrupts.

         Check $FRKHD.  Anything in Fork Queue?  No, go to 1.

         Remove entry from Fork Queue and update Fork Queue listhead
         pointers.

         Allow interrupts.

         If memory management and loadable drivers are defined, save
         APR5 and map the driver.
```

Restore fork context (registers R4 and R5).

Restore APR5 if memory management is defined.

Call routine whose fork context was restored (CALL @-(R3)).

Go to $DIRXT.

1. Is rescheduling required ($RQSCH not=0)? No, restore registers R0-R3 and go to 2.

   Allow interrupts.

   Is the power failure flag ($PWRFL) set? Yes, go to 3.

   Clear $RQSCH.

   Save context of current task.

   Locate a ready-to-run task.

   Load and check context of new task.

   Map windows of new task for correct mapping determined by task privilege.

   Go to $DIRXT.

2. Restore task stack pointer (from @$HEADR to SP).

   Increment stack depth indicator, $STKDP.

   Restore R4 and R5 from user stack and RTI.

3. Call power failure processing (CALL $POWER).

   Go to $DIRXT.

Notes:

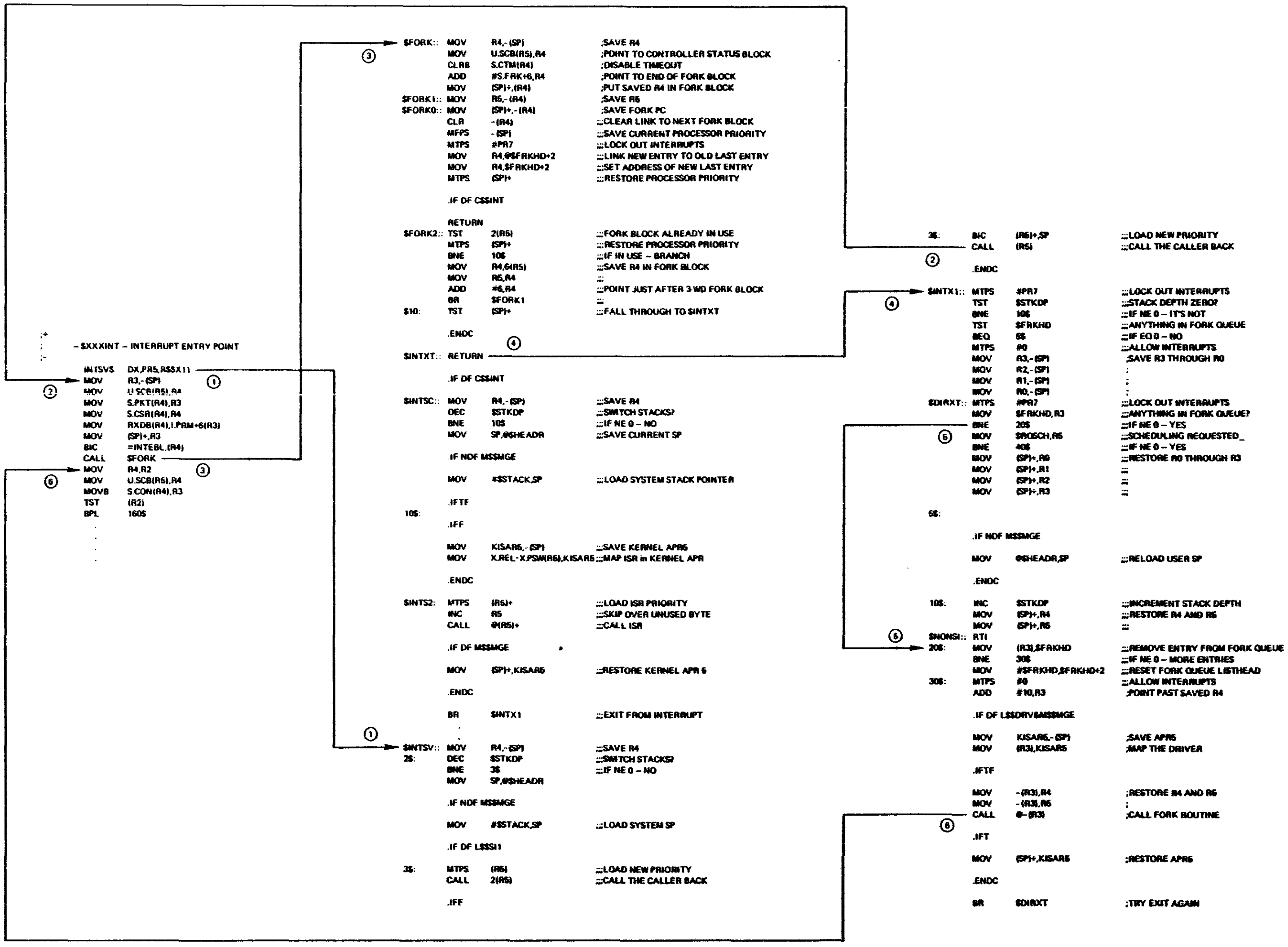
$DIRXT calls both waiting fork processes and the powerfail routine. These routines terminate via an RTS instruction. On return $DIRXT again cycles looking for work.

The task reschedule pointer, $RQSCH, controls the redispatching of the processor. It points to the location in the STD list where $DIRXT should begin its scan for a task ready to use the processor.

$RQSCH is set when a change of state has occurred in the system that might cause a task other than the one currently in control to obtain processor time. Examples are I/O done, clock queue runout, or a task doing an EXIT. The word is reset by $DIRXT just prior to its dispatching a new task.

3.3.10 Interrupt Processing Code

Figure 3-6 shows the driver and system interrupt code that is used in processing interrupts. The lines in Figure 3-6 shows the flow of control from routine to routine. The numbers associated with the lines indicate the sequence of events.



```
;+
;       - $XXXINT - INTERRUPT ENTRY POINT
;-
        INTSV$  DX,PR5,R$$X11
        MOV     R3,-(SP)
        MOV     U.SCB(R5),R4
        MOV     S.PKT(R4),R3
        MOV     S.CSR(R4),R4
        MOV     RXDB(R4),I.PRM+6(R3)
        MOV     (SP)+,R3
        BIC     =INTEBL,(R4)
        CALL    $FORK
        MOV     R4,R2
        MOV     U.SCB(R5),R4
        MOVB    S.CON(R4),R3
        TST     (R2)
        BPL     160$
          .
          .
          .
          .
```

```
$FORK::  MOV     R4,-(SP)                ;SAVE R4
         MOV     U.SCB(R5),R4            ;POINT TO CONTROLLER STATUS BLOCK
         CLRB    S.CTM(R4)               ;DISABLE TIMEOUT
         ADD     #S.FRK+6,R4             ;POINT TO END OF FORK BLOCK
         MOV     (SP)+,(R4)              ;PUT SAVED R4 IN FORK BLOCK
$FORK1:: MOV     R5,-(R4)                ;SAVE R5
$FORK0:: MOV     (SP)+,-(R4)             ;SAVE FORK PC
         CLR     -(R4)                   ;;;CLEAR LINK TO NEXT FORK BLOCK
         MFPS    -(SP)                   ;;;SAVE CURRENT PROCESSOR PRIORITY
         MTPS    #PR7                    ;;;LOCK OUT INTERRUPTS
         MOV     R4,@$FRKHD+2            ;;;LINK NEW ENTRY TO OLD LAST ENTRY
         MOV     R4,$FRKHD+2             ;;;SET ADDRESS OF NEW LAST ENTRY
         MTPS    (SP)+                   ;;;RESTORE PROCESSOR PRIORITY

         .IF DF C$$INT

         RETURN
$FORK2:: TST     2(R5)                   ;;;FORK BLOCK ALREADY IN USE
         MTPS    (SP)+                   ;;;RESTORE PROCESSOR PRIORITY
         BNE     10$                     ;;;IF IN USE - BRANCH
         MOV     R4,6(R5)                ;;;SAVE R4 IN FORK BLOCK
         MOV     R5,R4                   ;;;
         ADD     #6,R4                   ;;;POINT JUST AFTER 3-WD FORK BLOCK
         BR      $FORK1                  ;;;
$10:     TST     (SP)+                   ;;;FALL THROUGH TO $INTXT

         .ENDC

$INTXT:: RETURN

         .IF DF C$$INT

$INTSC:: MOV     R4,-(SP)                ;;;SAVE R4
         DEC     $STKDP                  ;;;SWITCH STACKS?
         BNE     10$                     ;;;IF NE 0 - NO
         MOV     SP,@$HEADR              ;;;SAVE CURRENT SP

         .IF NDF M$$MGE

         MOV     #$STACK,SP              ;;;LOAD SYSTEM STACK POINTER

         .IFTF
10$:
         .IFF

         MOV     KISAR6,-(SP)            ;;;SAVE KERNEL APR6
         MOV     X.REL-X.PSW(R5),KISAR5 ;;;MAP ISR in KERNEL APR

         .ENDC

$INTS2:  MTPS    (R5)+                   ;;;LOAD ISR PRIORITY
         INC     R5                      ;;;SKIP OVER UNUSED BYTE
         CALL    @(R5)+                  ;;;CALL ISR

         .IF DF M$$MGE

         MOV     (SP)+,KISAR5            ;;;RESTORE KERNEL APR 5

         .ENDC

         BR      $INTX1                  ;;;EXIT FROM INTERRUPT
            .
            .
$INTSV:: MOV     R4,-(SP)                ;;;SAVE R4
2$:      DEC     $STKDP                  ;;;SWITCH STACKS?
         BNE     3$                      ;;;IF NE 0 - NO
         MOV     SP,@$HEADR

         .IF NDF M$$MGE

         MOV     #$STACK,SP              ;;;LOAD SYSTEM SP

         .IF DF L$$SI1

3$:      MTPS    (R6)                    ;;;LOAD NEW PRIORITY
         CALL    2(R6)                   ;;;CALL THE CALLER BACK

         .IFF
```

```
3$:      BIC     (R6)+,SP                ;;;LOAD NEW PRIORITY
         CALL    (R6)                    ;;;CALL THE CALLER BACK

         .ENDC

$INTX1:: MTPS    #PR7                    ;;;LOCK OUT INTERRUPTS
         TST     $STKDP                  ;;;STACK DEPTH ZERO?
         BNE     10$                     ;;;IF NE 0 - IT'S NOT
         TST     $FRKHD                  ;;;ANYTHING IN FORK QUEUE
         BEQ     6$                      ;;;IF EQ 0 - NO
         MTPS    #0                      ;;;ALLOW INTERRUPTS
         MOV     R3,-(SP)                 ;SAVE R3 THROUGH R0
         MOV     R2,-(SP)                 ;
         MOV     R1,-(SP)                 ;
         MOV     R0,-(SP)                 ;
$DIRXT:: MTPS    #PR7                    ;;;LOCK OUT INTERRUPTS
         MOV     $FRKHD,R3               ;;;ANYTHING IN FORK QUEUE?
         BNE     20$                     ;;;IF NE 0 - YES
         MOV     $RQSCH,R5               ;;;SCHEDULING REQUESTED_
         BNE     40$                     ;;;IF NE 0 - YES
         MOV     (SP)+,R0                ;;;RESTORE R0 THROUGH R3
         MOV     (SP)+,R1                ;;;
         MOV     (SP)+,R2                ;;;
         MOV     (SP)+,R3                ;;;

6$:

         .IF NDF M$$MGE

         MOV     @$HEADR,SP              ;;;RELOAD USER SP

         .ENDC

10$:     INC     $STKDP                  ;;;INCREMENT STACK DEPTH
         MOV     (SP)+,R4                ;;;RESTORE R4 AND R5
         MOV     (SP)+,R5                ;;;
$NONSI:: RTI
20$:     MOV     (R3),$FRKHD             ;;;REMOVE ENTRY FROM FORK QUEUE
         BNE     30$                     ;;;IF NE 0 - MORE ENTRIES
         MOV     #$FRKHD,$FRKHD+2        ;;;RESET FORK QUEUE LISTHEAD
30$:     MTPS    #0                      ;;;ALLOW INTERRUPTS
         ADD     #10,R3                   ;POINT PAST SAVED R4

         .IF DF L$$DRV&M$$MGE

         MOV     KISAR6,-(SP)             ;SAVE APR6
         MOV     (R3),KISAR6              ;MAP THE DRIVER

         .IFTF

         MOV     -(R3),R4                 ;RESTORE R4 AND R5
         MOV     -(R3),R5                 ;
         CALL    @-(R3)                   ;CALL FORK ROUTINE

         .IFT

         MOV     (SP)+,KISAR6             ;RESTORE APR6

         .ENDC

         BR      $DIRXT                   ;TRY EXIT AGAIN
```

Figure 3-6 Interrupt Flow of Control

3.3.11 Interrupt Processing Summary

The following seven types of routines not only comprise the interrupt system, but practically comprise the entire Executive itself:

External Interrupt Routines

Trap Routines

Interrupt Save

Directive Save

Fork and Fork Processes

Interrupt Exit

Directive Exit

External interrupts cause traps to external interrupt processing routines which run in one of three states:

1. Non-interruptable at PR7.
   They run here when initially entered.

2. Interruptable by priorities higher than the interrupting source.

   Both states 1 and 2 are linearized being queued and dequeued from the system stack.

3. Fully interruptable as fork processes.

Trap routines, of which only one may occupy the system stack during any given passage through the Executive, operate at priority level zero, need never call Fork, and operate entirely from the system stack.

Interrupt save is called by driver interrupt routines when they make a transition from non-interruptable to interruptable.

Directive save is called by trap routines.

Fork creates a fork process for external interrupt routines.

Interrupt Exit unstacks waiting routines from the system stack, and when the system stack is empty drops into Directive Exit.

Directive Exit gives control to waiting fork processes, processes power failure, and redispatches the processor.

The Executive structure has a sequentiality which obviates the need for any explicit synchronizing mechanisms. Privileged tasks that follow the internal conventions of the Executive are never concerned with multiple-update of shared system data bases. While progressing toward the idle state the Executive gives priority to routines on the system stack, then to fork processes.

CHAPTER 4

PRIVILEGED TASKS

4.1 INTRODUCTION

This chapter discusses privileged tasks: what they are, the hazards to the system that they present, and how they are mapped.

4.2 PRIVILEGED TASKS

A privileged task has a special access to memory locations and devices that a nonprivileged task does not have. Specifically, certain privileged tasks can examine and use the values in system control blocks. These tasks can also examine and use Executive code. Also, certain privileged tasks can directly access the device registers in the I/O page. A privileged task can read from or write to a volume whether or not that volume is mounted (via the Mount command) or allocated to another user. These abilities imply that a privileged task has every ability that the Executive has, and, in fact, it does. It may be helpful to conceive of the privileged task as part of the Executive because certain privileged tasks are mapped with the Executive and I/O page. See Task Mapping in this chapter. As the writer of a privileged task, you are obliged to take every precaution so as not to damage the Executive, system, or user code and data.

4.2.1 Privileged Task Hazards

Privileged tasks are potentially hazardous to a running system. A privileged task can corrupt the system and disable devices. Bugs in these tasks are obscure and difficult to find. For these reasons, you must be cautious when developing and running a privileged task.

Make certain that your privileged task has completed its operation when you log off the system (say "BYE"). BYE does not abort privileged tasks as it does nonprivileged tasks. BYE cannot abort a privileged task because the privileged task may be in the process of changing the system data base. Therefore, it must be allowed to complete its processing. Also, if the privileged task is in system-state, BYE or no other task can execute until the privileged task completes its processing while in system-state. However, when the privileged task leaves system-state, BYE runs and logs you off the system, leaving the privileged task still in operation.

If a processor trap occurs in a privileged task while the task is in user-state, the Executive aborts the task. However, if the processor trap occurs in the privileged task while the task is in system-state, the system crashes. However, even while in user state the privileged task that is mapped to the Executive can cause a system crash by incorrectly changing system data. Please note that a privileged task in user-state should not be modifying system data.

Note that all tasks in an unmapped system can access all of memory. The privileged or nonprivileged designation has no particular meaning in an unmapped system. However, be just as careful about modifying Executive, device, or user data in an unmapped system.

4.2.2 Specifying a Task as Privileged

To specify a task as privileged, you must use the /PR switch in the Task Builder command line when you build your task. The RSX-11M Task Builder Reference Manual describes the use of this switch. You can use one of three numeric arguments with the /PR switch: 0, 4, or 5 (specifically as /PR:0, /PR:4, or /PR:5). The abilities and mapping of the privileged tasks designated by these switch values are described next.

4.2.2.1 /PR:0 Privileged Task

Using the /PR:0 switch causes the Task Builder to reserve user APAR 0 for mapping the task, which is the same as any other task. Virtual address space begins at virtual address 0 and extends upwards as far as 32K minus 32 words. This task cannot access the Executive routines, system data structures, or directly access the I/O page because the Task Builder has not reserved APRs for these purposes.

However, a task mapped into APR 0 can access the I/O page through a device common. The RSX-11M Task Builder Reference Manual discusses device commons. To do this, you must build a device common that occupies physical addresses on the I/O page. Then, when the task is built, you associate the common with the task by using the COMMON= or RESCOM= keyword.

There are four advantages to using a /PR:0 task and having it mapped into APR 0:

1. The task has more virtual address space available. A task mapped through APR 0 that accesses the I/O page can be as large as 28K words.

2. A device common provides you with the means to associate symbolic names with physical addresses in the I/O page.

3. You can restrict the amount of space to which the task has access on the I/O page. When you specify an argument of either 4 or 5 on the /PR switch, the Task Builder always allocates the entire I/O page. However, a device common can be as small as 32 words - expanding upwards (in 32-word blocks) to 4K words. Therefore, your task needs access to only the portion of the I/O page that it requires. Thus, there is less chance that the task will alter the wrong data and destroy the running system.

4. A /PR:0 task can write logical block I/O to a physically mounted volume, regardless of who issued the Mount or Allocate command. For example, the VMR task is a /PR:0 task and writes to mounted volumes during the SYSGEN process. However, this advantage can be hazardous for obvious reasons.

A /PR:0 task runs in user state and cannot switch to system state. Also, a /PR:0 task is not mapped to the Executive. If you want to write a privileged task that does I/O processing, it is to your advantage to use the /PR:0 switch for your task because there is less chance of corrupting the Executive or system code and data.

4.2.2.2 /PR:4 Privileged Task

If you want your privileged task to map to the Executive and I/O page, and your Executive is 16K or less, use the /PR:4 switch in the Task Builder command line. If you specify /PR:4 for your task, the Task Builder reserves APR 7 to map the I/O page and reserves APRs 0 through 3 to map the Executive as part of your task's virtual address space. The /PR:4 switch can be used only if your Executive size is 16K or less because the 16K Executive uses APRs 0 through 3 and your task is assigned mapping that starts with APR 4. Therefore, the Task Builder applies a bias of 100000 (16K decimal) to all virtual addresses within the task. This specific mapping of APRs 0 through 4 and 7 occurs whether the task is in user- or system-state.

There is up to 12K words of virtual address space possible in a /PR:4 task. The beginning of the task marks the end of the Executive code. If the task is 12K words in size, the end of the task marks the start of the I/O page. If the task is going to access the I/O page through APR 7, the task cannot exceed the 12K limit. If the task does exceed the limit, the Task Builder is forced to assign APR 7 to the task code. When building the task, the Task Builder does not give you an error message if your task exceeds the 12K limit. However, when you install the task, the system task, Install, sends you the following message:

```
"INS -- WARNING -- PRIVILEGED TASK OVERMAPS THE I/O PAGE"
```

This message is a warning that your task will most likely hang if it tries to access the I/O page.

A /PR:4 task can access all of the Executive, system control blocks, and I/O page. It can use Executive routines and do logical block I/O to a volume that is physically mounted on a device. Also, the /PR:4 task can issue a $SWSTK macro to change from user- to system-state. This allows the task to access the Executive or system code without interruptions or fear of the data being changed while it is being accessed. See $SWSTK in an Unmapped System or $SWSTK in a Mapped System in this chapter.

4.2.2.3 /PR:5 Privileged Task

If you want your privileged task to map to the Executive and I/O page, and your Executive is between 16K and 20K, use the /PR:5 switch in the Task Builder command line. If you specify /PR:5 for your task, the Task Builder reserves APR 7 to map the I/O page and reserves APRs 0 through 4 to map the Executive as part of your task's virtual address space. The /PR:5 switch can be used only if your Executive size is between 16K and 20K because the 20K Executive uses APRs 0 through 4 and your task is assigned APR 5. (APR 5 may be used if the Executive is less than 16K, but this wastes virtual address space.) Therefore, the Task Builder applies a bias of 120000 (20K) to all virtual addresses within the task. This specific mapping of APRs 0 through 5 and 7 occurs whether the task is in user- or system-state.

There is up to 8K words of virtual address space (12K if the I/O page is overmapped) possible in a /PR:5 task. The beginning of the task marks the end of the Executive code. If the task is 8K words in size, the end of the task marks the start of the I/O page. If the task is going to access the I/O page through APR 7, the task cannot exceed the 8K limit. If the task does exceed the limit, the Task Builder is forced to assign APR 7 to the task code. When building the task, the Task Builder does not give you an error message if your task exceeds the 8K limit. However, when you install the task, the system task, Install, sends you the following message:

"INS -- WARNING -- PRIVILEGED TASK OVERMAPS THE I/O PAGE"

This message is a warning that your task will most likely hang if it tries to access the I/O page.

A /PR:5 task can access all of the Executive, system control blocks, and I/O page. It can use Executive routines and do logical block I/O to a volume that is physically mounted on a device. Also, the /PR:5 task can issue a $SWSTK macro to change from user- to system-state. This allows the task to access the Executive or system code without interruptions or fear of the data being changed while it is being accessed. See $SWSTK in an Unmapped System or $SWSTK in a Mapped System in this chapter.

4.2.3 Writing a Privileged Task

In addition to the privileged task mapping and cautions mentioned previously, take note of the following points when writing a privileged task:

1. Task size is limited to 8K if you have a 20K Executive and 12K if you have a 16K Executive. This limit occurs because only two APRs are available for mapping your task with a 20K Executive and three APRs with a 16K Executive.

2. Your privileged task is mapped with the Executive and I/O page. This mapping is done to allow your task to access the Executive and I/O page registers (APRs, device registers, etc.). You can refer to any area in the Executive or I/O page by label or label and offset because you task build your task with the Executive symbol table file and library. A typical Task Builder command file is, for example:

   ```
   PRIV/PR:5,PRIV/-SP=PRIV
   [1,54]RSX11M.STB,[1,1]EXELIB/LB
   /
   UNITS=1              ;DEFINE NUMBER OF LUNS
   GBLDEF=OUTLUN:1      ;DEFINE LUN ON WHICH TO PRINT MESSAGES
   ASG=TI0:1            ;ASSIGN LUN TO DEVICE
   ```

   In this command file, the Task Builder is informed that the task has a privilege attribute of 5. Therefore, the task uses APR 5. It also uses APR 6 if it is over 4K in length.

3. When you use a privileged task, the Executive has dedicated almost all the APRs to the necessary mapping for the Executive, the I/O page, and your task. Your task can issue PLAS directives to remap any number of these APRs to regions. However, such remapping can cause obscure and difficult to find system bugs. Also, be aware that when a directive unmaps a window that formerly mapped the Executive or the I/O page, the Executive restores the former mapping.

4. A privileged task uses the $SWSTK macro when going from user- to system-state. (See $SWSTK in a Mapped System.) While in system state your task can access the Executive and I/O page without being interrupted by other tasks or system processes. It would be prudent to limit the time that your task spends in system state for the sake of other system users. Remember that while in user-state your task can not only read, but change the Executive or I/O page, if necessary. However, if your task is interrupted while changing data in user state, it may not finish its processing properly, thereby causing obscure bugs. Allowing a task to manipulate the Executive data base while in user-state is not a goal of RSX-11M. Future releases of RSX-11M may prohibit this activity.

5. While in system-state, a privileged task can modify any mapping registers to make them point to any desired area of physical memory. For example, the system task, PMD, loads the starting address of the task being dumped into KISAR6 (Executive APR 6). It then biases the addresses by 140000 to force mapping through APR 6. The bias is 120000 for APR 5. Note that by modifying APR 5 or APR 6 registers, it is possible for a privileged task to unmap its task image. Therefore, you must take care to avoid this. However, PMD is only 4K words in size. Therefore, modifying KISAR6 cannot cause PMD to unmap itself.

4.2.4 The $SWSTK Routine Described - Unmapped and Mapped Systems

4.2.4.1 SWSTK$ in a Mapped System

SWSTK$ - Used by privileged tasks to switch from user- to system-state (described for a mapped system).

1. TASK CODE

```
CALL SWSTK$,100$
      ...
      ...
      ...
      ...
      ...
      ...
    RETURN
  100$:...
```

EXPANDS TO:

```
 CALL $SWSTK,100$
  EMT 376
  .WORD 100$
```

PROCESS:
EMT puts current PS and PC on stack, loads PC from location 30 ($EMTRP) and PS from location 32 (PR7 - defined as 340). Current PC points to .WORD containing address of 100$:. EMT in a mapped system causes switch to system SP.

SYSTEM STACK

TASK PC
TASK PS

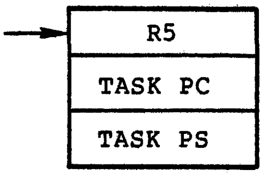
2. DRDSP

```
$EMTRP::DIRSV$
        TST $STKDP
```

PROCESS EMT 376:

EXPANDS TO: JSR R5, $DIRSV Puts R5 on stack. R5 then contains return address of the TST instruction. Jumps to $DIRSV::

SYSTEM STACK



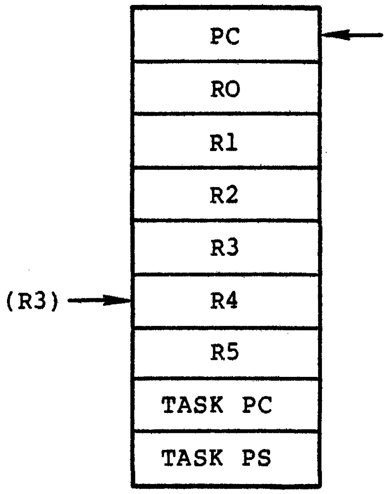
3. SYSXT

```
$DIRSV::MOV R4,-(SP)
        DEC $STKDP
            ...
        MOV SP,@$HEADR
        MTPS #0
        MOV R3,-(SP)
        MOV R2,-(SP)
        MOV R1,-(SP)
        MOV R0,-(SP)
        CALL (R5)
        BR $DIRXT
```

PROCESS:

Save R4 on stack. Decrement $STKDP. Stack depth: user- or system-state; 1=USER, 0 or less = SYSTEM. Save current SP (->). MTPS expands to:
  CLRB @#PS
  Clears first byte of PS to allow interrupts.
Save R3 through R0 on stack.
CALL (R5) expands:
  JSR PC,(R5)
  Puts PC on stack. Puts return address in R5 into PC. Jumps to TST following $EMTRP::.

SYSTEM STACK



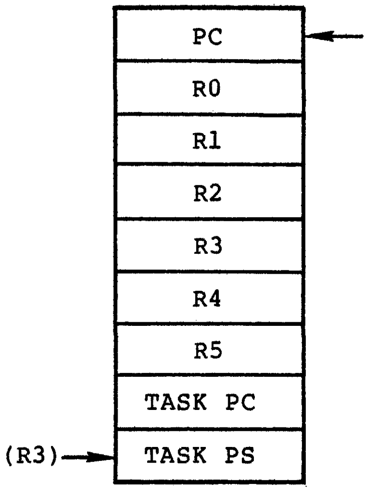
4. DRDSP

```
$EMTRP::DIRSV$
    --->TST $STKDP
        BNE 70$
        MOV @$HEADR,R3
        CMP (R3)+,(R3)+
    *   MOV (R3)+,R5
        MFPI -(R5)
        CMP #104377,(SP)
        BNE 80$
        .
        .
    80$:CMP (R5),#104376
        JMP $SWSTK
    85$:JMP $EMSST
```

PROCESS:

Check for 0 $STKDP. If not 0, crash. Get saved task SP into R3. Point to task PC in stack. Get address of word after $SWSTK into R5. (PC saved on stack after $SWSTK in task. Back up R5 to check EMT. Check EMT. If EMT 376, jump to $SWSTK. If not, process SST.

SYSTEM STACK



* Task stack pointer (R3) shown at this point.

5. SYSXT

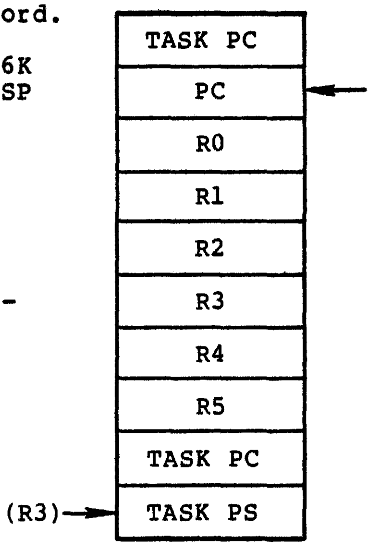
```
   $SWSTK:CLRB (R3)
          MOV #KISAR6,R5
          MOV UISAR6,(R5)
          MOV UISAR5,-(R5)
(16K cond)MOV UISAR4,-(R5)
          MOV -(R3),R5
          MOV (R5)+,(R3)
          MOV R5,(SP)
          MOV -(R3),R5
          MOV 12(SP),R3
          CALLR @(SP)+
```

PROCESS:

Clear byte 0 of task's PS word. Move user APRs to Executive APRs. Move user APR 4 if 16K Executive. Get saved task SP for PC. Put PC for return to user task in R3. Put task PC on system stack. Restore R5. Restore R3.

CALLR expands:
JMP @(SP)+
Return to task instruction after $SWSTK to process system-state code.

SYSTEM STACK



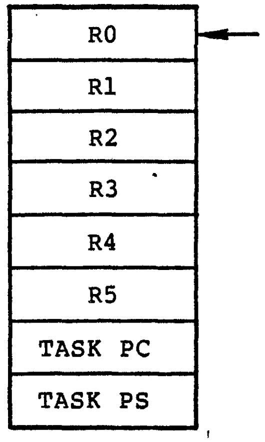
6. TASK CODE

```
 CALL SWSTK$,100$
——►...
     ...
    RETURN
 100$:
```

TASK PROCESS:

Do system-state processing. RETURN expands:
RTS PC
Pops stack and gets saved PC to go back to SYSXT at BR $DIRXT instruction.

SYSTEM STACK



7. SYSXT

PROCESS:

```
        BR $DIRXT
        ...
        ...
        ...
        ...
$DIRXT::MTPS #PR7
        MOV $FRKHD,R3
        BNE  --
        MOV $RQSCH, R5
        BNE  --
        MOV (SP)+,R0
                .
                .
        MOV (SP)+,R3
        INC $STKDP
        MOV (SP)+,R4
        MOV (SP)+,R5
        RTI
```

Lock out interrupts.
Check forking.
Check rescheduling.
Restore R0 - R3.
Make $STKDP = 1.
Restore R4, R5.
RTI pops saved PC and PS. from stack.

8. TASK CODE

```
SAVED PC ———>.WORD 100$
         100$: code
```

PROCESS:

Process at 100$ in user-state.

4.2.4.2 SWSTK$ in an Unmapped System

SWSTK$ - Used by privileged tasks to switch from user- to system-state (described for an unmapped system)

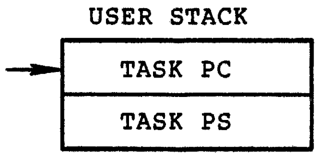
1. TASK CODE

```
CALL SWSTK$,100$
      ...
      ...
      ...
      ...
      ...
      ...
    RETURN
 100$:...
```

EXPANDS TO:

```
CALL $SWSTK,100$
EMT 376
.WORD 100$
```

PROCESS:
EMT puts current PS and PC on stack, loads PC from location 30 ($EMTRP) and PS from location 32 (PR7 - defined as 340). Current PC points to .WORD containing address of 100$:. Still using user SP at this point. Switch to system SP.

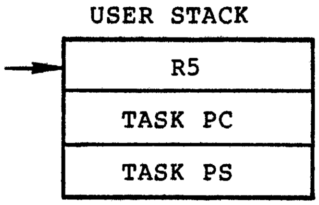


2. DRDSP

```
$EMTRP::DIRSV$
        TST $STKDP
```

PROCESS EMT 376:

EXPANDS TO:
JSR R5, $DIRSV
Puts R5 on stack. R5 then contains return address of the TST instruction. Jumps to $DIRSV::

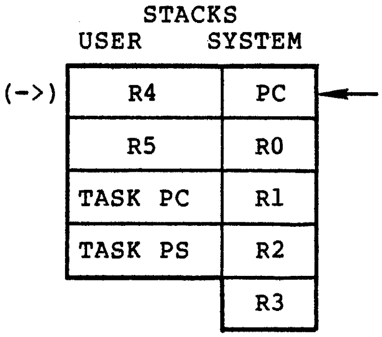


3. SYSXT

```
$DIRSV::MOV R4,-(SP)
        DEC $STKDP
            ...
        MOV SP,@$HEADR
        MOV #$STACK,SP
        MTPS #0
        MOV R3,-(SP)
        MOV R2,-(SP)
        MOV R1,-(SP)
        MOV R0,-(SP)
        CALL (R5)
     *  BR $DIRXT
```

PROCESS:

Save R4 on stack.
Decrement $STKDP.
Stack depth: user- or system-state; 1=USER, 0 or less = SYSTEM. Save current SP (->). Load system SP.
MTPS expands to:
  CLRB @#PS
  Clears first byte of PS
  to allow interrupts.
Save R3 through R0 on system stack.
CALL (R5) expands:
  JSR PC,(R5)
  Puts PC for return
  to BR $DIRXT (*) for system
  exit. Puts return address
  in R5 into PC. Jumps to
  TST following $EMTRP::.

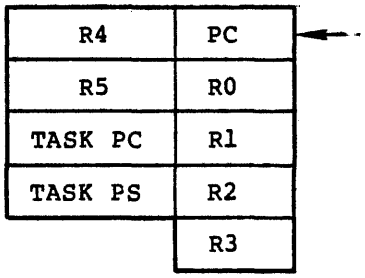


```
4.  DRDSP                        PROCESS:

  $EMTRP::DIRSV$
     ——►TST $STKDP           Check for 0 $STKDP.
        BNE 70$               If not 0, crash.
        MOV @$HEADR,R3        Get saved task SP into R3.
        CMP (R3)+,(R3)+       Point to task PC in stack.
    *   MOV (R3)+,R5          Get address of word after
        CMP #104377,-(R5)     $SWSTK into R5.  PC      (R3)->
        BNE 80$:              saved on stack after $SWSTK
                              in task.  Back up R5 to
    80$:CMP (R5),#104376      check EMT. Check EMT.
        JMP $SWSTK            If EMT 376, jump to $SWSTK.
    85$:JMP  EMSST            If not, process SST.

    * User stack pointer (R3) shown at this point
```

STACKS



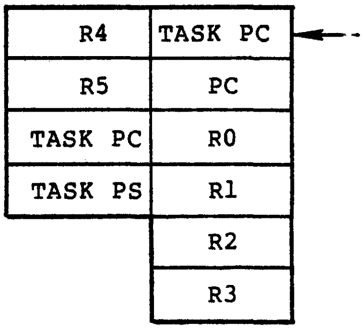
USER	SYSTEM
R4	PC
R5	R0
TASK PC	R1
TASK PS	R2
	R3

```
5.  SYSXT                        PROCESS:

   $SWSTK:CLRB (R3)           Clear byte 0 of task's PS
        MOV -(R3),R5          word. Get saved task SP
        MOV (R5)+,(R3)        for PC. Put PC for return
        MOV R5,-(SP)          to user task in R3.  Put task
        MOV -(R3),R5          PC on system stack.  Restore
    *   MOV 12(SP),R3         R5. Restore R3.  CALLR
        CALLR @(SP)+          expands:  JMP @(SP)+
                                 Return to task to
                                 instruction after
                                 $SWSTK to process
                                 system-state code.

     * System stack pointer shown at this point
```

STACKS



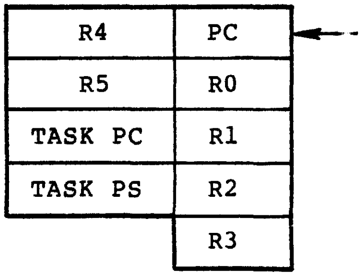
USER	SYSTEM
R4	TASK PC
R5	PC
TASK PC	R0
TASK PS	R1
	R2
	R3

```
6.  TASK CODE                    TASK PROCESS:

  CALL SWSTK$,100$            Do system-state processing
  ——►...                      RETURN expands:
       ...                        RTS PC
       ...                    Pops stack and gets
  *    RETURN                 saved PC to go back to
  100$:                       SYSXT at BR $DIRXT
                              instruction.

  * System SP shown at this point
```

STACKS



USER	SYSTEM
R4	PC
R5	R0
TASK PC	R1
TASK PS	R2
	R3

```
7.   SYSXT                        PROCESS:

          BR $DIRXT
                .
                .
                .                 Lock out interrupts.
   $DIRXT::MTPS #PR7              Check forking.
          MOV $FRKHD,R3           Check rescheduling.
          BNE  --                 Restore R0 - R3.
          MOV $RQSCH, R5          Restore R0 - R3.
          BNE  --                 Reload user SP.
          MOV (SP)+,R0            Make $STKDP = 1.
                .                 Restore R4, R5.
                .                 RTI pops saved PC and PS.
          MOV (SP)+,R3            from stack.
          MOV @$HEADR,SP
          INC $STKDP
          MOV (SP)+,R4
          MOV (SP)+,R5
          RTI

8. TASK CODE                      PROCESS:

   SAVED PC ——►.WORD 100$         Process at 100$ in
           100$: code             user-state
```

4.2.5 Task Mapping

Figure 4-1 shows the mapping of a nonprivileged user task in an unmapped system.

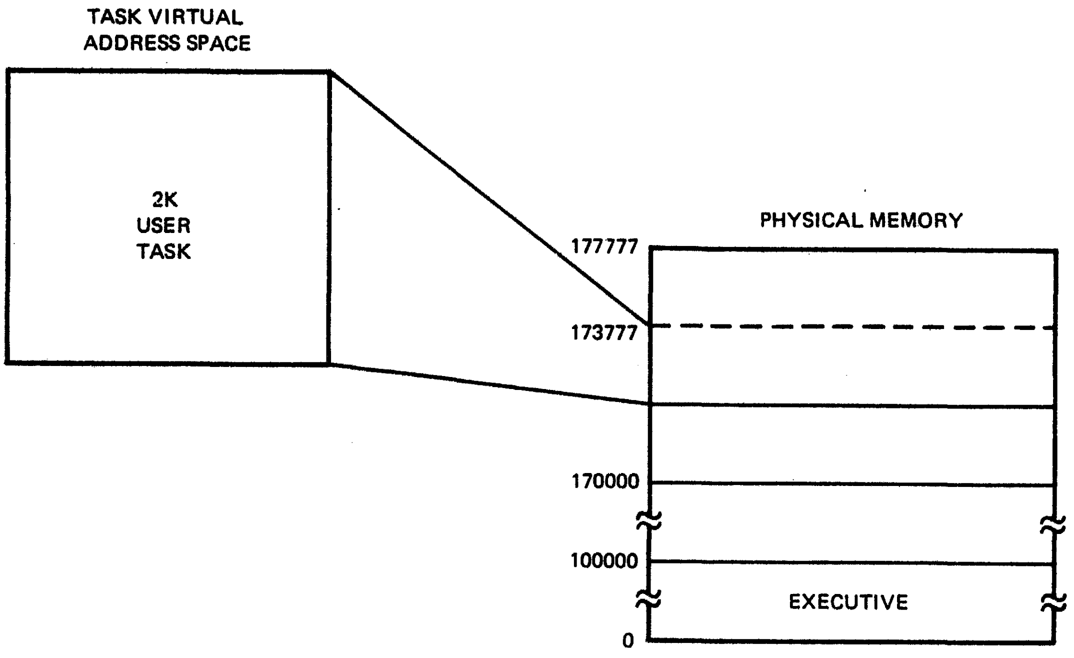
Figure 4-2 shows the mapping of a nonprivileged task in a mapped system.

Figure 4-3 shows the mapping of an 8K nonprivileged task in a mapped system.

Figure 4-4 shows the mapping of an 8K nonprivileged task that uses memory management (PLAS) directives in a mapped system.

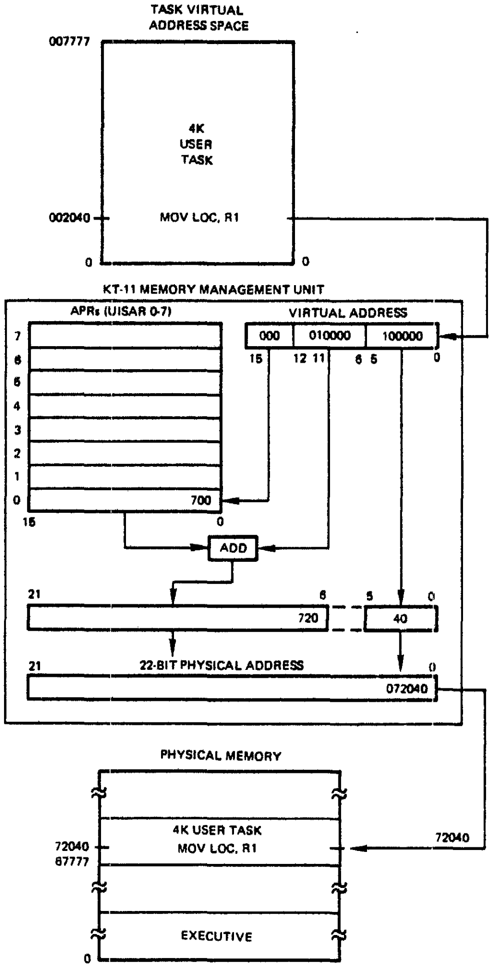
Figure 4-5 shows an 8K privileged task mapped into APRs 5 and 6. The 20K Executive occupies 5 APRs, which leaves two APRs for the privileged task and one for the I/O page. All the APRs are used by the system in this example. The Executive copies what are normally the Executive's own APRs (KISAR0 through KISAR4, and KISAR7) into what is normally the user APRS (UISAR0 through UISAR4, and UISAR7). Thus, the privileged task has access to the Executive and I/O page in either user- or system-state.

These figures are presented in a sequential and logical way to allow an easier understanding of privileged task mapping in the system.



- Location of task depends on partition location in unmapped system (PAR option of TKB)
- Base of task on 32-word boundary; length is a multiple of 32 words; highest virtual address = 28K
- Non-runable task — resident library or global common — maximum size = 32K minus 32 words
- Unless you change parameters of task and task build again, task in unmapped system always loads into the same location
- INSTALL calls $ALOCB to allocate TCB space. It then reads the first record of header label blocks, and sets up data in the task header. It verifies that the task fits in the specified partition
- INSTALL sets load device and LBN in TCB. UCB address of load device is put in header of INSTALL as a result of opening task image file. It then puts UCB address in TCB of task
- INSTALL: checks System Task Directory (STD) for task of same name; gets partition name from task Label Block Group; searches for task partition; checks if task fits in partition
- INSTALL: checks Label Block Group for PLAS support, resident overlays; puts task offset in TCB, checks if task is a common block, checkpoint file space, set priority in TCB, checks partition base address and task starting address
- INSTALL fills in TCB with information from Label Block Group. When everything is done, TCB is in STD
- When the task is executed, the Executive Loader loads task into correct partition and location. TCB is put on Active Task List

Figure 4-1 User Task in Unmapped System



- Nonprivileged user task uses User Active Page Register 0 (APR 0), which is called UISAR0 in the I/O page.

- User is aware of virtual addresses 0 - 07777_8 only.

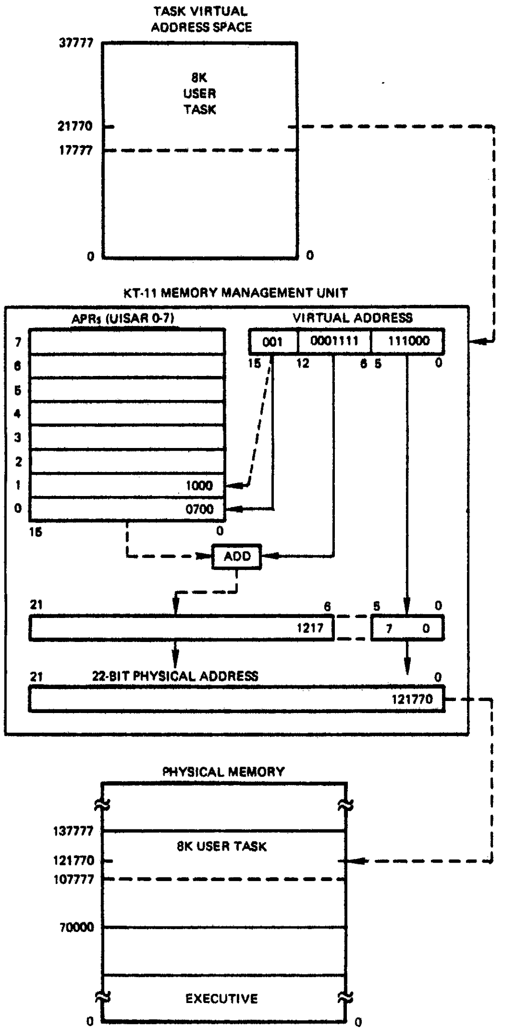
- INSTALL task maps this 4K task with APR 0 only, because task is only 4K. If it were bigger, INSTALL would select needed number of APRs (4K for each of 8 APRs — 32K maximum). INSTALL determines into which partition task is to go in mapped system and virtual address for base of task (task may have a virtual section at beginning of task code, making its real virtual address different from its apparent virtual address). Executive determines physical address by information in task header and current system memory allocation. Executive puts this address in APR0. After task is loaded, mapping information is kept in Executive copy of task header in DSR.

- Contents of APR 0 relocates whole task (MOV instruction shown as part of task as an example).

- Bits 13-15 of task virtual address selects APR 0. If task were 8K, two APRs would be used. Virtual addresses in the task (bits 13-15) select dynamically the APR from those assigned.

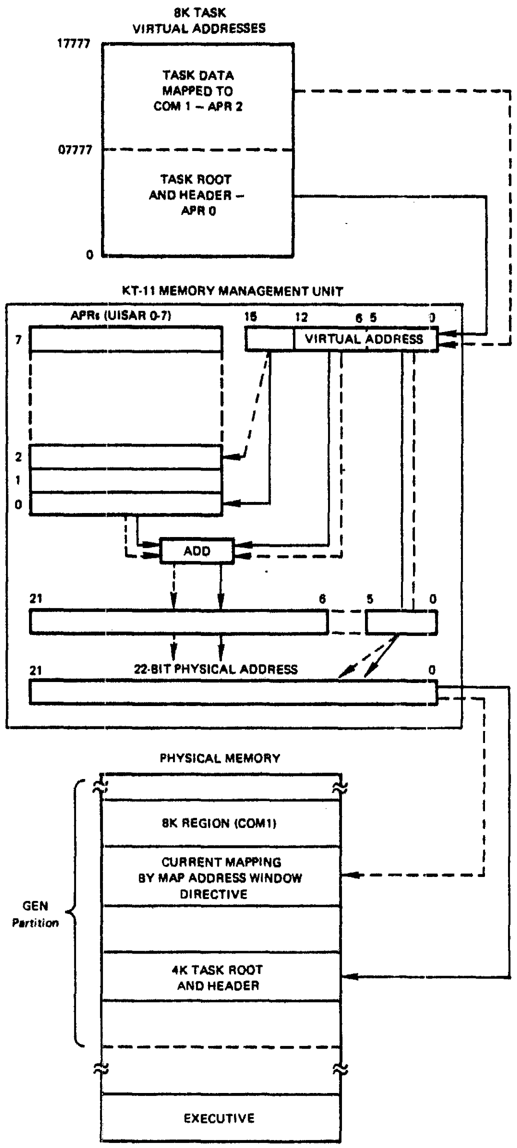
- KT-11 adds value in APR 0 (16 bits) to bits 6-12 of virtual address. KT-11 appends sum to the low-order 6 bits of the virtual address to produce 22-bit physical address.

Figure 4-2 4K Nonprivileged User Task Mapping in a PDP-11/70



- User task shown has no memory management directives.
- User task is 8K in size. Because each APAR maps 4K of memory, user task must use two APARs.
- For example, APR 1 relocates address 21770 in the user task to physical address 121770. Because the virtual address being translated is over 4K and less than 8K, the high-order three bits of the address are 001. The three high-order bits select APR 1. If the task were between 8K and 12K in size, virtual addresses between 8K and 12K would contain 2 in the high-order digit and APAR 2 would be selected for these addresses. This relationship of virtual addressing to APRs continues up to 32K of virtual addresses. Each APAR can contain a 4K relocation factor; therefore, with 8 APARs, 32K of virtual addresses can be relocated.
- The virtual address limit of a user task in a mapped system is 32K.
- Without memory management directives, user task without commons or resident libraries occupies continuous physical address space in memory. The task can only access that physical memory to which the Executive maps it. This physical memory is continuous and has a direct one-to-one relationship to the tasks's virtual addresses until the task exits or the Executive checkpoints it. Of course, the Executive does not move fixed tasks once they are loaded, but they may be shuffled.
- Executive memory allocation routines determine the physical base address of the user task when the Executive Loader loads the task into memory. The base address is likely to change if checkpointing or shuffling occurs.

Figure 4-3 8K Nonprivileged User Task Mapping in a PDP-11/70



- The Task Builder built this task with WNDWS option = 1; the Task Builder reserves two window blocks in Task Header. Window blocks contain mapping information (see Chapter 2). Window block 0 is for APR 0 mapping. TKB always reserves one window block (window 0) for mapping task header and root (in this case 0 to 7777). Window block 1 is for address window (APR 2) that PLAS directives use in the task. However, a task may use more than one address window for mapping when using PLAS directives, but the number of windows to be used must be specified to TKB when building the task.

- Upper 4K of this task is defined as an address window to access a region in memory.

- Window size can be 32 to 32K minus 32 words and must start on a 4K virtual boundary.

- The Executive maps both the task root and the region into the GEN system-controlled partition. You must specify the APR that the Executive is to use for the region.

- The Executive may map the region to a numerically lower physical location than your task and the region and task root may or may not be contiguous in memory.

- You must create Window Definition Block (WDB) in your task (see Chapter 2 or RSX-11M Executive Reference Manual).

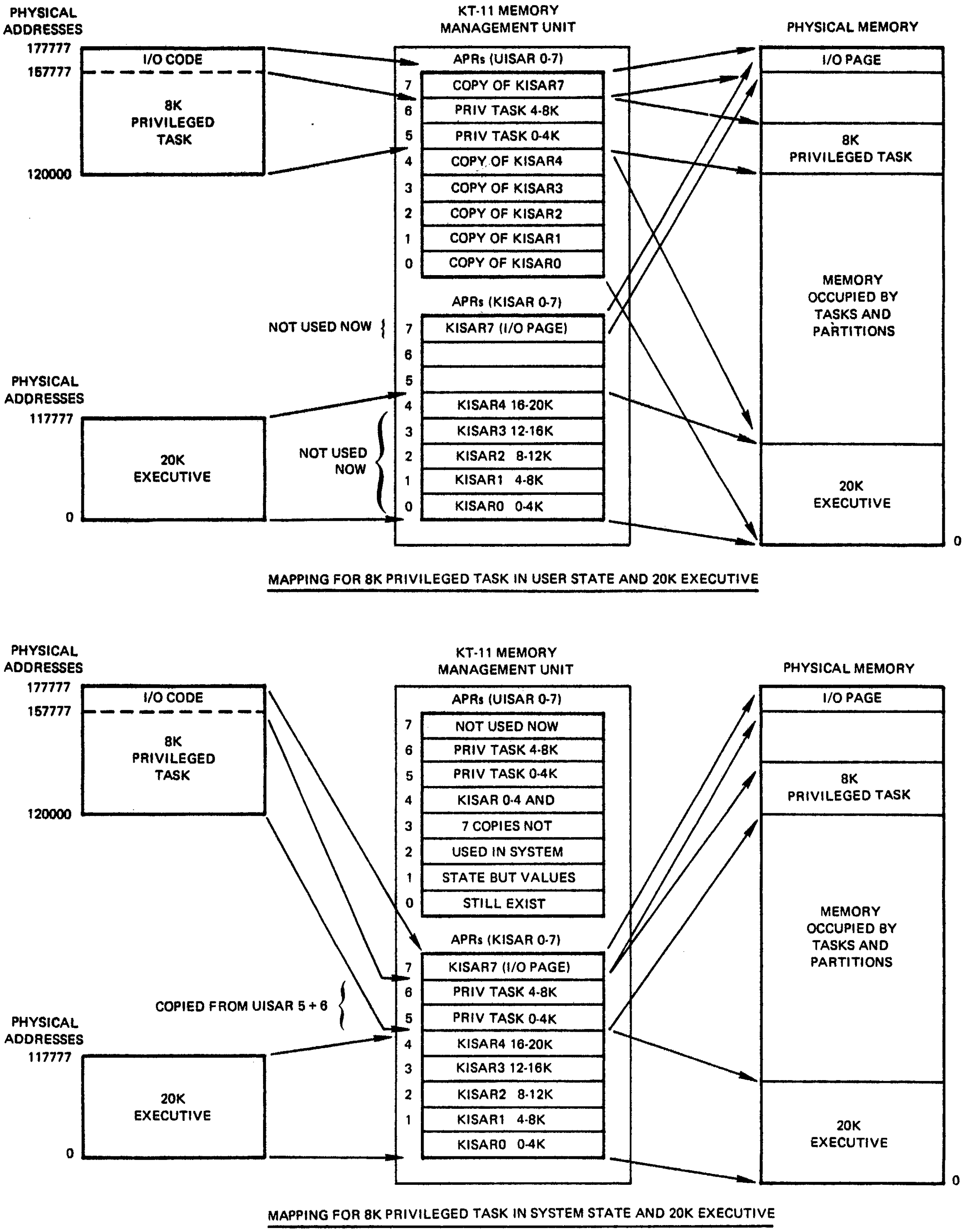
- You must create the Region Definition Block (RDB) in your task (see RSX-11M Executive Reference Manual).

- Your task must issue an Attach Region directive to attach the region (COM1) to your task.

- Your task must issue a Map Address Window directive to map your task's window to the region. In this directive, your task specifies the offset into the region in which mapping begins. This offset can be changed by issuing another Map Address Window with the offset changed. Doing this can move the window through the whole region.

- These directives in combination with the Send/Receive and Detach Region directives make it possible for two tasks to transfer data to each other through a region that they can both access in common.

Figure 4-4 8K Nonprivileged Task Mapping in a PDP-11/70 Using PLAS Directives



MAPPING FOR 8K PRIVILEGED TASK IN USER STATE AND 20K EXECUTIVE

MAPPING FOR 8K PRIVILEGED TASK IN SYSTEM STATE AND 20K EXECUTIVE

Figure 4-5 Privileged Task Mapping

CHAPTER 5

MCR INTERFACE

5.1 MCR - MONITOR CONSOLE ROUTINE

MCR is the collection of functions that make it possible to operate and control the RSX-11M system from a terminal device. As the link between the collection of services provided by RSX-11M and users who want to make use of these services, MCR provides a number of commands which you can execute by entering the command at your terminal.

The RSX-11M Operator's Procedures Manual describes all the MCR commands that you can use. They are listed as follows:

1. ABOrt
2. ACS
3. ACTive
4. ALLocate
5. ALTer
6. ASsigN
7. ATL
8. BRK
9. CANcel
10. CLQueue
11. DEAllocate
12. DEVices
13. FIX
14. HELP
15. LUN
16. OPEn register
17. PARtition definitions
18. REAssign
19. REDirect

20. REMove

21. RESume

22. RUN

23. SAVe

24. SET

25. TAsk List

26. TASk list

27. TIMe

28. UNFix

Services 29-39 run as tasks.

29. BOOt

30. BROadcast

31. BYE

32. DMOunt

33. HELlo

34. INTitvolume

35. INStall

36. LOAd

37. MOUnt

38. UFD

39. UNLoad

5.1.1 Structure and Operation Environment of MCR

MCR is an RSX-11 task that, typically, operates out of its own partition. MCR runs at a moderately high priority to be able to service terminal input requests. This priority value is typically 160. or, at least, higher than that of user tasks and utilities. MCR is privileged and is usually stopped, though active, while waiting for terminal input. MCR can be checkpointed and have checkpoint space in its own task image.

MCR is a tree structured task, and its structure is depicted schematically in Figure 5-1.

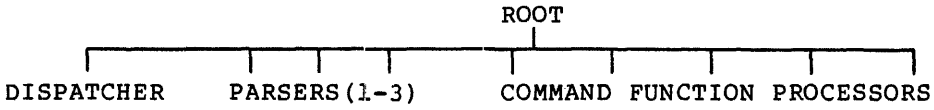


Figure 5-1 MCR Tree Structure

The command function processors are those that process the first 16 console services listed in Section X.X. The remaining console services run as tasks and not as integral parts of MCR. MCR, in fact, does not distinguish between these task functions and tasks that it initiates as a result of recognizing an MCR request for functions 17-22 listed in Section 5.1. The console language syntax is defined such that if the first three characters of an input line are not part of the defined command language, then MCR attempts to initiate the task named

```
...xxx
```

Thus, the task named ...JIM can be initiated by entering

```
JIM
```

to MCR, or by entering

```
RUN ...JIM
```

to MCR.

5.1.2 The Terminal Driver and MCR Initiation

The terminal driver is intimately integrated into the operation of MCR. Because RSX-11M accepts and acts upon unsolicited input from any operator terminal, it is the function of the terminal driver to know when it is receiving input destined for MCR.

When a character on an operator terminal is struck, the resulting interrupt initiates the terminal driver. (Remember the device is full duplex and the keyboard cannot be locked to prevent input when the device is, in fact, involved in an I/O operation.) The driver then acts on the input as follows:

1. [Check the device state]

   Is the device busy. No, go to 3.

2. [The device is busy]

   If the driver was sending output (in an output state) when the character was entered, an input request flag is set in the appropriate UCB and the driver continues sending the output stream. When the output request is finished, processing continues at 5.

   If the terminal was in an input state the character is accepted. Go to 6.

3. [Device is not busy]

   Note, if the device was not busy, the incoming character is the first character of an input line.

   Was the input character a CTRL-C? (CTRL-C is an explicit request to communicate with MCR.) If the character was a CTRL-C, the terminal driver executes a $FORK and execution continues at 4.

If the first character is not a CTRL-C, a check is made to see if the device is attached. If it is, MCR ignores the character (unsolicited input to MCR on an attached device is not permitted).

If the device is unattached then it is considered the beginning of unsolicited input to MCR. Go to 4.

4. [Fork level processing]

   The driver has transferred to fork level because it needs a buffer, and it can only get a buffer at fork level (shared system tables must be altered to obtain a buffer). In addition to getting a buffer, the fork level terminal processing code must check for a rare race condition.

   After the arrival of the CTRL-C (or a non CTRL-C character if the terminal is not attached) and between the time the fork is executed and control is regained in the driver, it is possible that the device may have returned to the busy state. This is because we may have just unbusied the device for a previous request when the input interrupt occurred. The interrupt code finds the device free and executes a fork. But before control is regained at fork level, execution is continued in the driver for the previous request. The driver jumps to the initiator entry to propagate its execution and thus may find another waiting I/O request which it begins processing because the device is free. Thus the fork routine must recheck the state of the device. If it is busy the input is ignored and the driver returns (exits) from fork level. Otherwise, an attempt is made to obtain a buffer for the unsolicited input.

5. [Buffer Acquisition]

   If the buffer acquisition attempt is unsuccessful, the driver ignores the input and exits.

   If a buffer is obtained, the driver sets up to start an unsolicited input request by initializing various pointers and setting the status of the controller and unit to busy.

   If the initial input character was CTRL-C, then

   MCR>

   is echoed to signify an explicit request to input to MCR.

   Otherwise, the input character is stored in the buffer and echoed on the initiating terminal.

   The driver returns (exits) from fork level.

6. [Character processing]

   Once the terminal driver determines that input coming from an operator terminal is destined for MCR, it transfers subsequent characters into the buffer acquired in Step 5. It also echoes the incoming characters. The acceptance of input ceases if:

   a. The buffer is filled (the buffer has room for 80 characters) but the maximum accepted depends on the device:

      72 for KSR

      72 for VT05B

      80 for LA30S

      80 for LA30S

   b. An end of line character is entered. The valid end of line characters are:

      CTRL-Z

      Carriage Return

      ALT-Mode (codes 33, 175, and 176)

7. [Interrupt from a character echo]

   Is the device in input mode? If it is not, another character is obtained from the user output buffer and it is echoed. If the device is in input mode, end-of-line must be checked. If it is not, the keyboard interrupt is re-enabled and exit from the interrupt occurs. If end-of-line is detected, then the fork process is called.

8. [End-of-line processing - fork level]

   For unsolicited input, the UCB address and the terminating character are deposited into the input buffer and the buffer linked into MCR's input queue. MCR is then requested to run. The driver itself clears control and unit busy and returns to its initiator entry point.

   For solicited input, I/O Done is called. First, the number of characters entered is determined and the buffered input is moved to the soliciting task's input buffer. The driver input buffer is released and I/O Done is called with the second I/O status word equal to the number of bytes entered. The left byte of the first I/O status word is set equal to the terminating character and the right byte to +1. The driver then jumps to the initiator entry point to propagate its execution.

5.1.3 MCR Operation

After the request of MCR by the driver, the file system is swapped out and MCR is swapped in. Control is passed to the MCR root segment which calls the Dispatcher (DSPTCH) overlay. DSPTCH, via a privileged subroutine ($SWSTK), switches to system state. The call to this routine includes a parameter which is the address where the caller wants to return when it switches back to task state. The state switching routine performs the switch and resumes processing in the caller immediately following the call. When $SWSTK is called, it sets up an interrupt entry to the system. Interrupts are locked out while it pushes the passed return address and the PS on the stack. $SWSTK then calls interrupt save ($INTSV). On return from interrupt save, R3, R2, R1, R0 are pushed onto the stack and now the stack state simulates that of an EMT. $SWSTK now calls the caller who resumes execution one instruction past the call to $SWSTK. When the calling routine finishes, it returns, which takes it back to $SWSTK. $SWSTK jumps to Directive Exit which redispatches the processor. The effect of this is to resume the caller in task state at the passed return address.

MCR now proceeds as follows:

1. [Request an unsolicited input queue entry]

   The Dispatcher calls the queue removal routine ($QRMVF). $QRMVF attempts to remove a buffer and deliver it to the Dispatcher. If no buffer is available (carry set return from $QRMVF) the Dispatcher exits. The buffer is formatted as shown in Figure 5-2.

◄─ WORD ─►	◄─ WORD ─►	◄─ UP TO 80 BYTES ─►
LINK TO NEXT BUFFER	UCB OF INPUT DEV	COMMAND INPUT

Figure 5-2 Input Buffer

   The queue empty condition never occurs on an initial call to MCR, because MCR is not requested unless something is in the queue. MCR remains resident until it has processed all the entries in the unsolicited input queue.

   Note that the Dispatcher, during buffer requisition, is operating at system level and all queue entries are done at fork level. Thus the buffer removal process is linearized with buffer item entry.

   If DSPTCH gets a buffer, it saves the buffer address in a memory location and does a return. This return takes DSPTCH back to task state where the processing of the buffer begins.

2. [Process a Buffer]

   On return to task state, the Dispatcher scans through the buffer and

   1. Compresses out redundant spaces and/or tabs

   2. Converts an Escape character to a Carriage Return

   3. Truncates trailing spaces and/or tabs

If no line terminator is found in the buffer, the Dispatcher inserts a CR as the 80th character. Finally the actual line terminator (either CR or ESC) is saved so it can be restored in the message if the message must be routed to a task other than MCR itself.

The Dispatcher then converts the first three characters to RAD50 and begins to search two internal tables for an MCR function with a matching name.

3. [Searching the function tables - Table descriptions]

MCR has two function tables; one for privileged commands, and one for non-privileged commands.

Privileged commands are those whose unrestricted use could cause privacy violation or system failures and they can only be executed from a privileged terminal. Privileged terminals are identified by a bit in the UCB. These terminals are established at SYSGEN or by the SET command.

Both tables contain a 5-word packet for each command in the class (privileged or non-privileged). The packet appears in Figure 5-3.

RAD50 CMD NAME (3 CHARS)
INDEX INTO COMMAND OVERLAY
ADDRESS OF PARSER TABLE
RAD50 COMMON OVERLAY NAME
INDEX INTO COMMON OVERLAY

Figure 5-3 Function Table Entry

The table entries correspond to three overlay types:

1. Command overlay

2. Parser overlay

3. Common overlay

The use of these overlay types is noted next.

Typically, any command that can be processed in a single overlay and whose size is such that it requires all or nearly all of the max overlay size (600 wds) is classed as a command overlay.

Parsers for the commands are distinct entities and are grouped into overlays. Generally, a given parser services more than one command but three parsers currently service all the commands. The parser entry is a pointer to a parser table entry shown in Figure 5-4.

RAD50 PARSER NAME (3-CHARS)
INDEX INTO PARSER OVERLAY

Figure 5-4 Parser Table Entry

Because three parsers service all the commands it is more economical in storage space to point to the parser table than to include the name and the index in the main function table.

The index is used as the entry point into the parser where the parsing for a given command begins. This is required because a parser can, and generally does, contain parsers for more than one command.

The common overlay is used when the processing for a command is small enough to make it practical to group more than one command into a single overlay. This grouping saves space since ten words are required by the Overlay Runtime System for each overlay in a tree structure. The index serves the same purpose as the index in a parser overlay.

Note that a command overlay also contains an index. The value of the command overlay index is generally zero. But to maintain the coherence of the table processing commonality, and to allow for flexibility, the index is included.

3a. [Look up and start a function other than an MCR internal function]

The Dispatcher then looks in the privileged command table for a name which matches the first three characters in the input buffer. This table contains all the privileged MCR commands.

Internally, privileged terminals are identified by a bit in the UCB. The bit is set at SYSGEN or from a privileged terminal using the SET MCR command.

If the command is not found in the privileged command table, the non-privileged command table is searched.

MCR for multiuser-systems is called MCRMU. MCRMU is actually two tasks: ...MCR and ...SYS. The Dispatcher in MCRMU, which is part of the MCR task, looks in a table to see if it is to process the issued command. These commands are: LUN, REA, REM, FIX, UNFIX, CANCEL, RESUME, and ABORT. If the command is not in this table, the Dispatcher looks in another table to see if the task, ...SYS, processes the command.

If the name is not in either table, then the Dispatcher prefixes three periods to the three buffer characters, and using these six characters, searches the STD looking for a match on the name. If it does not find the name it displays an error message on the initiating terminal. If it finds the name, it requests the function to run, supplying as an argument to the requested task the UCB address that was in the input buffer. The UCB address is inserted into the TCB of the requested task as its TI (terminal input) pseudo device. If the attempt to request the task fails, an error message is displayed, the buffer is released, and MCR exits. Having discovered a non-internal MCR function, MCR must prepare to pass the buffer, because the initiated task is going to issue a Get MCR Command Line directive. To pass the buffer, MCR uses three words in system common. These words are:

1. The TCB address of the requested task
2. The address of the command buffer
3. The byte count of the number of input characters in the buffer

MCR fills these words, making synchronizing checks that they are free, because only one triplet exists for passing buffers to a requested task. Thus, until the buffer is emptied, other completed buffers in the queue are waiting.

Eventually, the requested task starts running, and issues a Get MCR Command Line directive. The directive processing then tests for a match on the TCB address in SYSCM and the TCB address of the requesting task. If they match the buffer is passed to the task by copying it into the DPB of the directive. The directive status is set to the byte count, the buffer is released and the TCB address in the SYSCM triplet is cleared. The TCB address being zero is an indication to MCR that the triplet is free.

3b. [Start an internal MCR function]

Once a name match has been found in the command table, the Dispatcher copies words 1, 2, 4 and 5 of the function table entry and both words of the parser table entry for this command into the MCR root segment. Now the Dispatcher scans the function table entry as follows:

3c. If a parser exists, go to 4. Otherwise, if a command overlay exists go to 3d. If a command overlay does not exist, go to 5. Otherwise, abort the system.

3d. The Dispatcher forms the overlay name, constructs the required overlay information packet, and enters the root at the point where overlay loading is performed.

4. [Parser functions]

The selected parser parses the buffer and, if the parse is successful, it jumps back to the root to load the desired function. If the parse fails, the parser deposits an error number in the root and jumps to the entry $ERLD in the root which loads the error overlay.

Ultimately the root initiates another routine, either the error routine or the requested function.

5. [Function routines]

These routines may further check the input and find errors. If errors are found, the function sets up the error routine and jumps to the root to load an error overlay. If it succeeds, the function releases the buffer and enters the root as the point where the root reloads the dispatcher.

6. [Error Overlay]

   The error overlay contains all error messages and the code needed to format the error message from the error number deposited in the root by the MCR component discovering the error.

7. [Final Exit]

   The dispatcher calls the queue routine to obtain another buffer; if one is found, the cycle of name table scanning resumes (starting at Step 2). If no buffers are waiting, MCR exits.

CHAPTER 6

I/O PROCESSING

This chapter contains a description of QIO directive processing in the form of a Logical Flow Diagram.

6.1 IMPLEMENTATION

The user interface to the RSX-11M I/O system consists of logical unit numbers (LUNs) and a single active I/O directive, Queue I/O. (The directives Assign LUN, Get LUN Info, etc. do not initiate I/O transfers.)

In RSX-11M all the preliminary processing antecedent to actually queuing an I/O request is performed by the QIO directive processing code that uses the I/O data structures. This code calls ancillary routines for centralized services. When a driver finally receives an I/O order, it generally has very little to do other than set up the status registers and issue the order.

Termination processing is equally modular and centralized. The driver is entered, performs some cleanup operations, and calls centralized routines for obtaining pending I/O orders, performing AST processing, etc. The driver is only concerned with the most intimate and specific details of the actual hardware interface in respect to the execution and completion of I/O transfers. Using this centralization philosophy, RSX-11M keeps both driver size and non-interruptible processing time small.

6.2 RSX-11M I/O DATA STRUCTURES

The static I/O data structures consist of three blocks:

1. A Device Control Block (DCB)

2. A Unit Control Block (UCB)

3. A Status Control Block (SCB)

Although each serves a specific function, and the components of each, in general, reflect these functions. The functional purpose of each data structure is reflected by the units of information which compose them. See Chapter 8, Data Areas and Control Blocks.

6.2.1 The Device Control Block (DCB)

One device control block exists for each device type attached to the system. Its function is to describe the static characteristics of both the controller and the units attached to the controller. All the DCBs in the system are singly linked. The DCB contains such information as:

- The device mnemonic (two ASCII characters)
- The lowest and highest unit numbers on the respective controller type
- The address of the first UCB
- The length of each UCB
- The next DCB pointer
- The Legal Function Mask
- The Control Function Mask
- The No-Op'd Function Mask
- The File Function Mask
- The pointer to the Driver Dispatch Table

For Loadable drivers, the pointer to the driver's driver dispatch table (D.DSP) and the pointer to the driver's PCB (D.PCB) are altered by Load and Unload.

The rest of these information fields are static and are used principally by the Queue I/O directive processing code to prepare a Queue I/O request for a device driver. The details of Queue I/O processing are shown in Figure 6-2, QIO Directive Processing.

6.2.2 The Unit Control Block (UCB)

One unit control block exists for each physical device unit attached to the system. Many of its information fields are static and very device dependent. The device independent parts of the UCB contain few dynamic parameters. For example, the redirect pointer reflects the result of a Redirect MCR command.

The UCB contains device unit specific data, such as unit status, physical unit number, and unit characteristics. See Chapter 8, Data Areas and Control Blocks.

6.2.3 The Status Control Block (SCB)

One status control block exists for each device controller in the system. This is true even if the controller handles more than one device unit (like the DK Controller). Line multiplexers such as the DH11 and DJ11 are considered to have a controller per line since all lines may transfer in parallel.

Some of the information in the SCB is dynamic. It contains the following information about the currently active unit:

- The interrupt vector address
- The controller bus request priority
- Timeout counts (initial and current)
- The address of the Control Status Register
- The address of the current I/O Packet
- Storage for a Fork Block
- The I/O queue listhead
- The controller status (busy/idle)
- The controller index

The dynamic data in the SCB makes it possible to maintain control of the current I/O in progress on the controller. The presence of Fork Block storage in the SCB implies that a driver cannot call $FORK twice for the processing at fork level on a given controller. The driver for a specific device type never concerns itself with unwanted recursion or multiple updates. Once a driver is in a fork level process, further I/O processing, which may involve updating a shared data base, is automatically locked out by the Fork processing of the system itself.

6.3 QUEUE I/O DIRECTIVE PARAMETER BLOCK

The Queue I/O directive requires a 12-word Directive Parameter Block (DPB) as shown in Figure 6-1.

LENGTH	DIC
FUNCT CODE	MODIFIER
RESERVED	LUN
PRIORITY	EFN
I/O STATUS BLOCK ADDRESS	
ADDRESS OF AST ROUTINE	
DEVICE DEPENDENT PARAMETERS	

Figure 6-1 Queue Directive Parameter Block

The parameters have the following interpretation.

DPB size (required):

The length of DPB in words. For QIO always equal to 12 words.

DIC (required):

Directive Identification Code. For QIO, the value is a 1.

Function Code and Modifier (required):

The code of the requested I/O function (0 through 31) and device dependent modifier bits. I/O function code definitions are in the RSX-11M I/O Drivers Reference Manual.

Reserved:

Reserved byte; must not be used.

LUN (required):

Logical Unit Number.

Priority:

Request priority. Ignored by RSX-11M, but space must be allocated for RSX-11D compatibility.

EFN (optional):

Event flag number.

I/O Status Block Address (optional):

This word contains a pointer to the I/O status block, which is a 4-byte device dependent I/O completion data packet formatted as:

Byte 0

I/O status byte

Byte 1

Augmented data supplied by the driver

Bytes 2 and 3

The contents of these bytes depend on the value of byte 0. See the RSX-11M I/O Drivers Reference Manual.

AST Address (optional):

Address of an AST service routine.

Device Dependent Parameters:

Up to six parameters specific to the device. These may be, for example:

Buffer address

Byte count

Carriage control type

Logical block number

Any optional parameters that are not specified must be filled with zeros.

Directive Parameter Blocks and their contents are fully described in the RSX-11M Executive Reference Manual and the RSX-11M I/O Drivers Reference Manual.

6.4 QIO DIRECTIVE LOGICAL FLOW

Figure 6-2 is a Logical Flow Diagram of QIO directive processing. The diagram shows the main stream of code flow through the DRQIO module of the Executive.

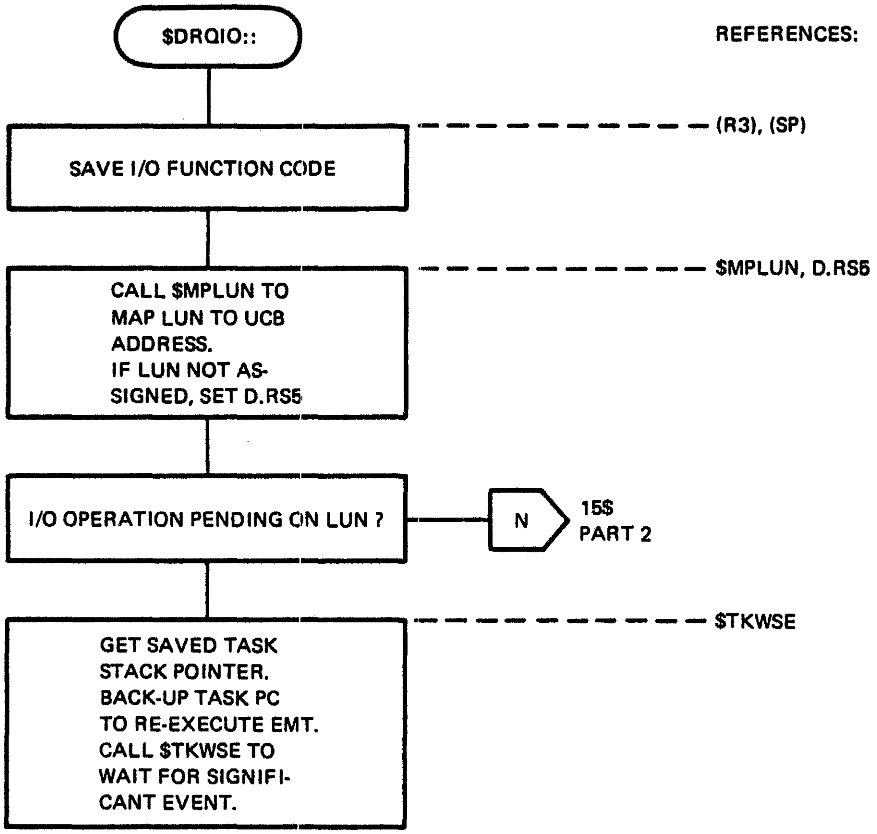

$DRQIO::
REFERENCES:
(R3), (SP)
SAVE I/O FUNCTION CODE
$MPLUN, D.RS5
CALL $MPLUN TO MAP LUN TO UCB ADDRESS. IF LUN NOT ASSIGNED, SET D.RS5
I/O OPERATION PENDING ON LUN ?
N
15$ PART 2
$TKWSE
GET SAVED TASK STACK POINTER. BACK-UP TASK PC TO RE-EXECUTE EMT. CALL $TKWSE TO WAIT FOR SIGNIFICANT EVENT.

Figure 6-2 QIO Directive Processing (Part 1 of 24)

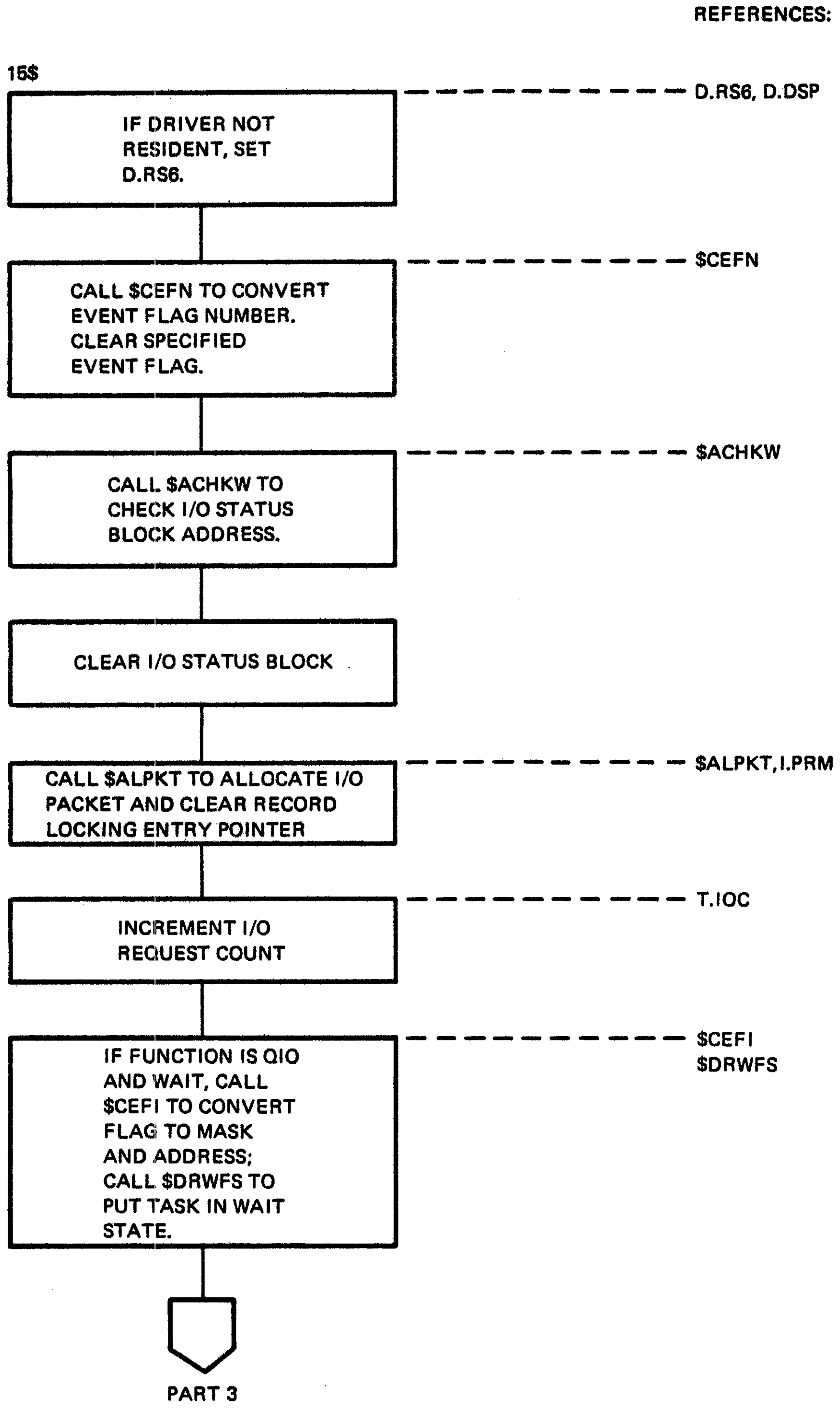

REFERENCES:
15$
IF DRIVER NOT RESIDENT, SET D.RS6.
D.RS6, D.DSP
CALL $CEFN TO CONVERT EVENT FLAG NUMBER. CLEAR SPECIFIED EVENT FLAG.
$CEFN
CALL $ACHKW TO CHECK I/O STATUS BLOCK ADDRESS.
$ACHKW
CLEAR I/O STATUS BLOCK
CALL $ALPKT TO ALLOCATE I/O PACKET AND CLEAR RECORD LOCKING ENTRY POINTER
$ALPKT,I.PRM
INCREMENT I/O REQUEST COUNT
T.IOC
IF FUNCTION IS QIO AND WAIT, CALL $CEFI TO CONVERT FLAG TO MASK AND ADDRESS; CALL $DRWFS TO PUT TASK IN WAIT STATE.
$CEFI
$DRWFS
PART 3

Figure 6-2 QIO Directive Processing (Part 2 of 24)

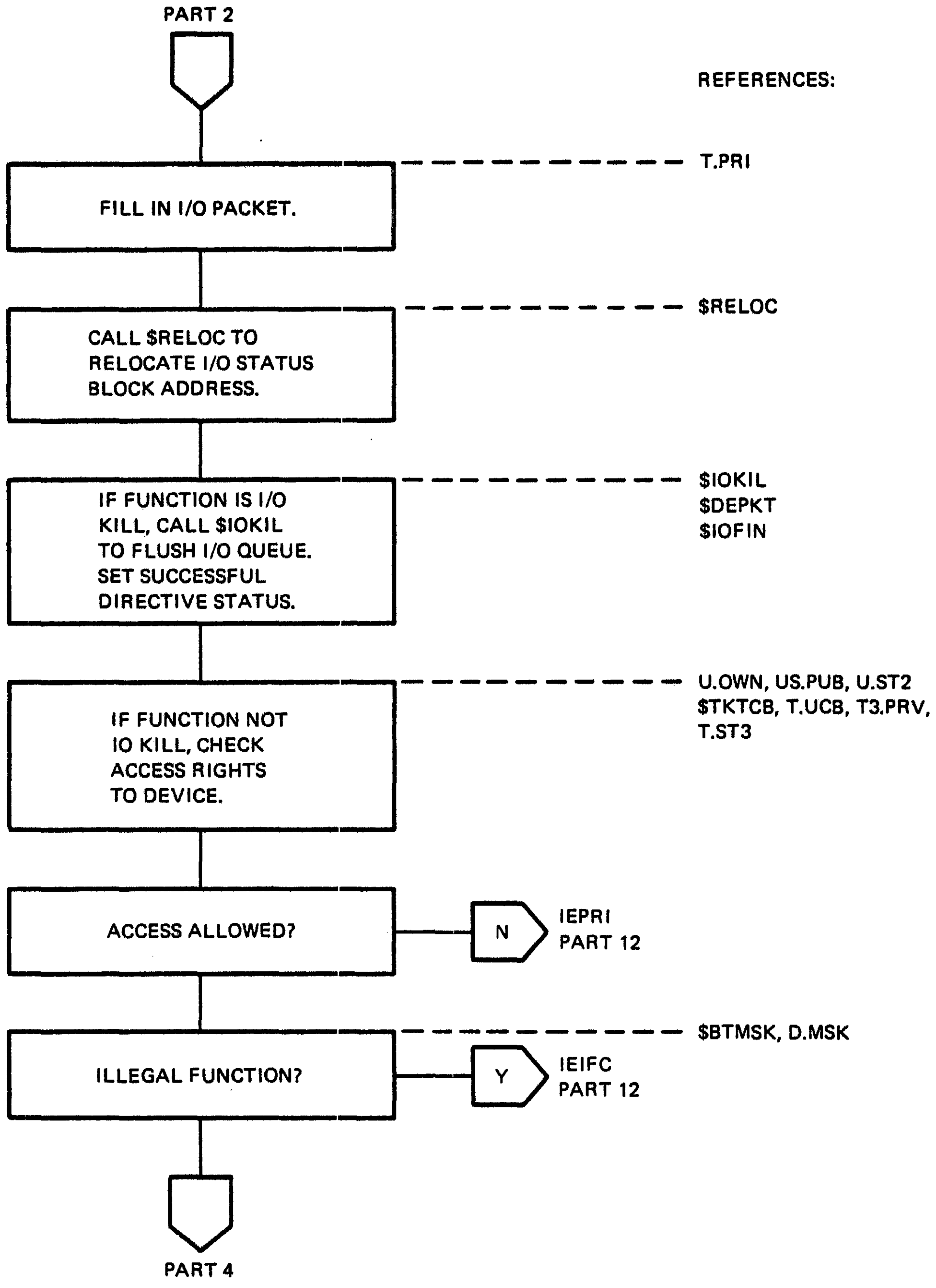

PART 2
REFERENCES:
T.PRI
FILL IN I/O PACKET.
$RELOC
CALL $RELOC TO RELOCATE I/O STATUS BLOCK ADDRESS.
$IOKIL
$DEPKT
$IOFIN
IF FUNCTION IS I/O KILL, CALL $IOKIL TO FLUSH I/O QUEUE. SET SUCCESSFUL DIRECTIVE STATUS.
U.OWN, US.PUB, U.ST2
$TKTCB, T.UCB, T3.PRV,
T.ST3
IF FUNCTION NOT IO KILL, CHECK ACCESS RIGHTS TO DEVICE.
ACCESS ALLOWED?
N
IEPRI
PART 12
$BTMSK, D.MSK
ILLEGAL FUNCTION?
Y
IEIFC
PART 12
PART 4

Figure 6-2 QIO Directive Processing (Part 3 of 24)

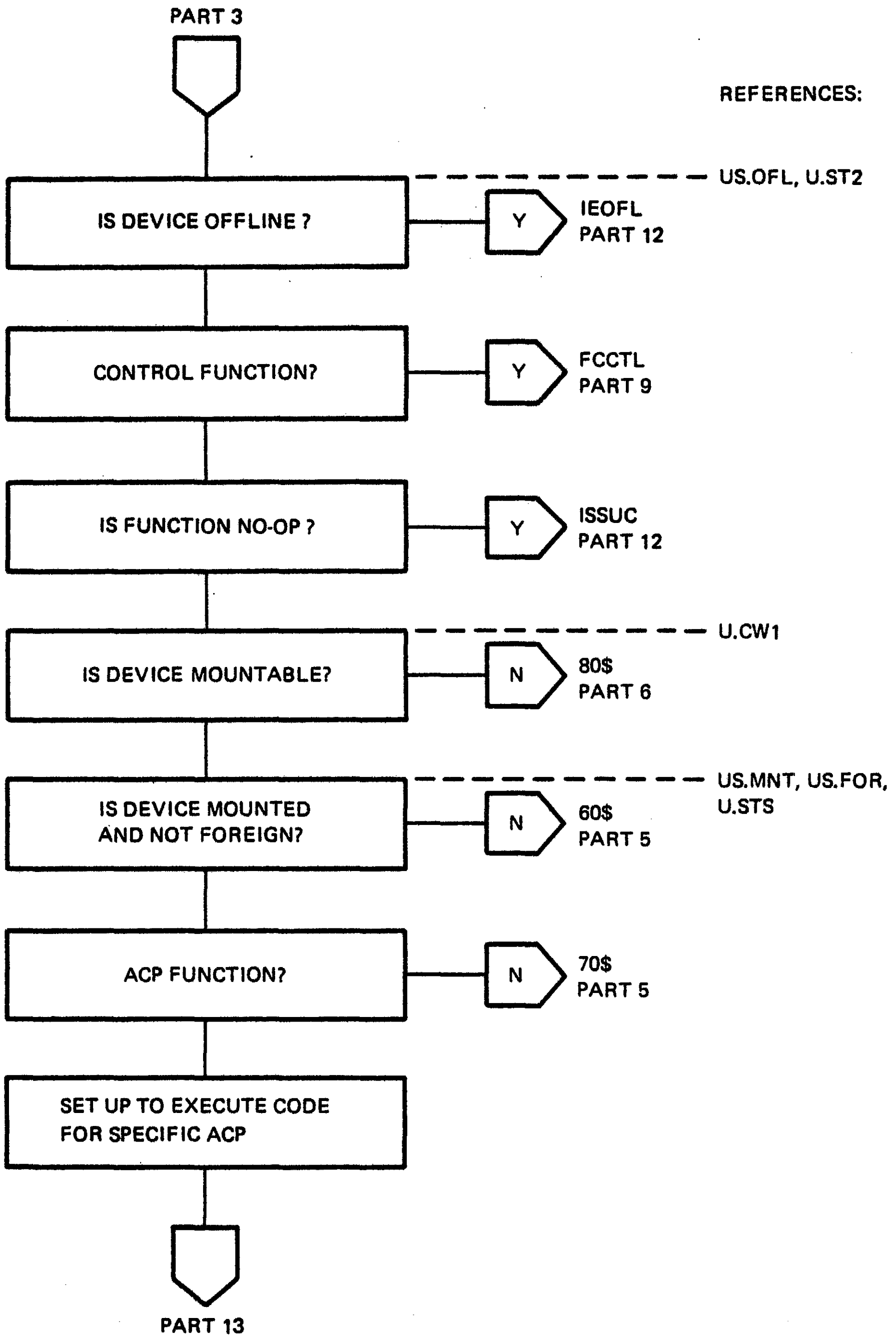


Figure 6-2 QIO Directive Processing (Part 4 of 24)

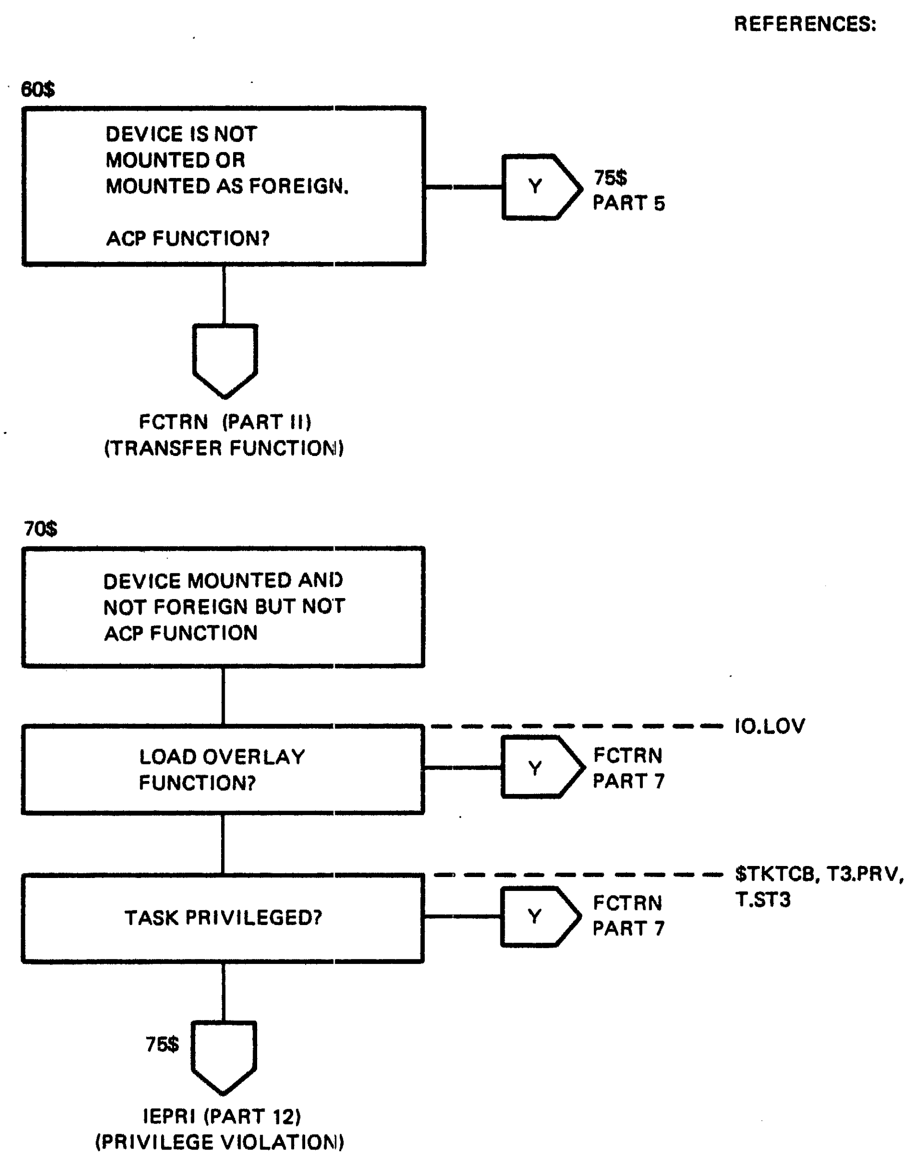


Figure 6-2 QIO Directive Processing (Part 5 of 24)

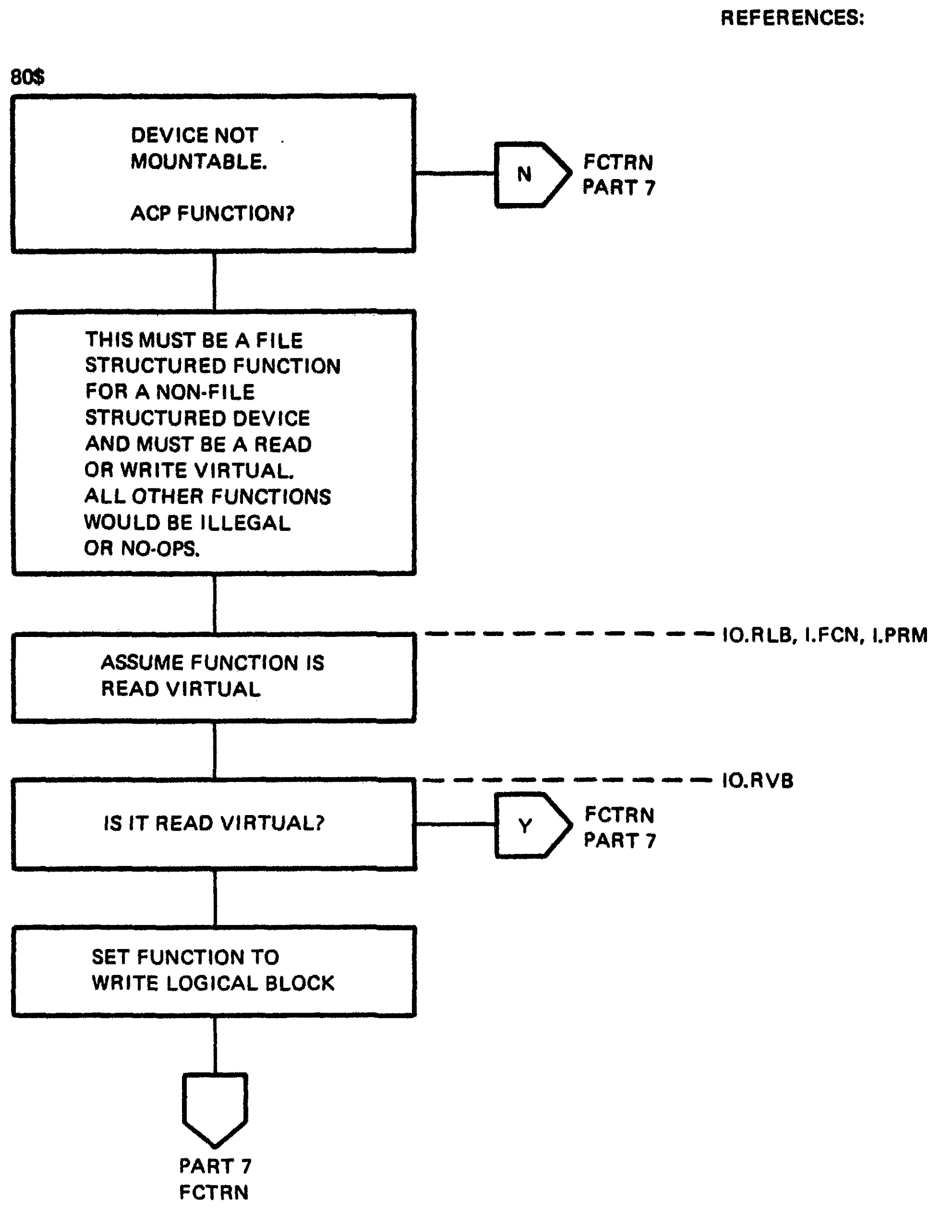

REFERENCES:
80$
DEVICE NOT MOUNTABLE.
ACP FUNCTION?
N
FCTRN PART 7
THIS MUST BE A FILE STRUCTURED FUNCTION FOR A NON-FILE STRUCTURED DEVICE AND MUST BE A READ OR WRITE VIRTUAL. ALL OTHER FUNCTIONS WOULD BE ILLEGAL OR NO-OPS.
IO.RLB, I.FCN, I.PRM
ASSUME FUNCTION IS READ VIRTUAL
IO.RVB
IS IT READ VIRTUAL?
Y
FCTRN PART 7
SET FUNCTION TO WRITE LOGICAL BLOCK
PART 7 FCTRN

Figure 6-2 QIO Directive Processing (Part 6 of 24)

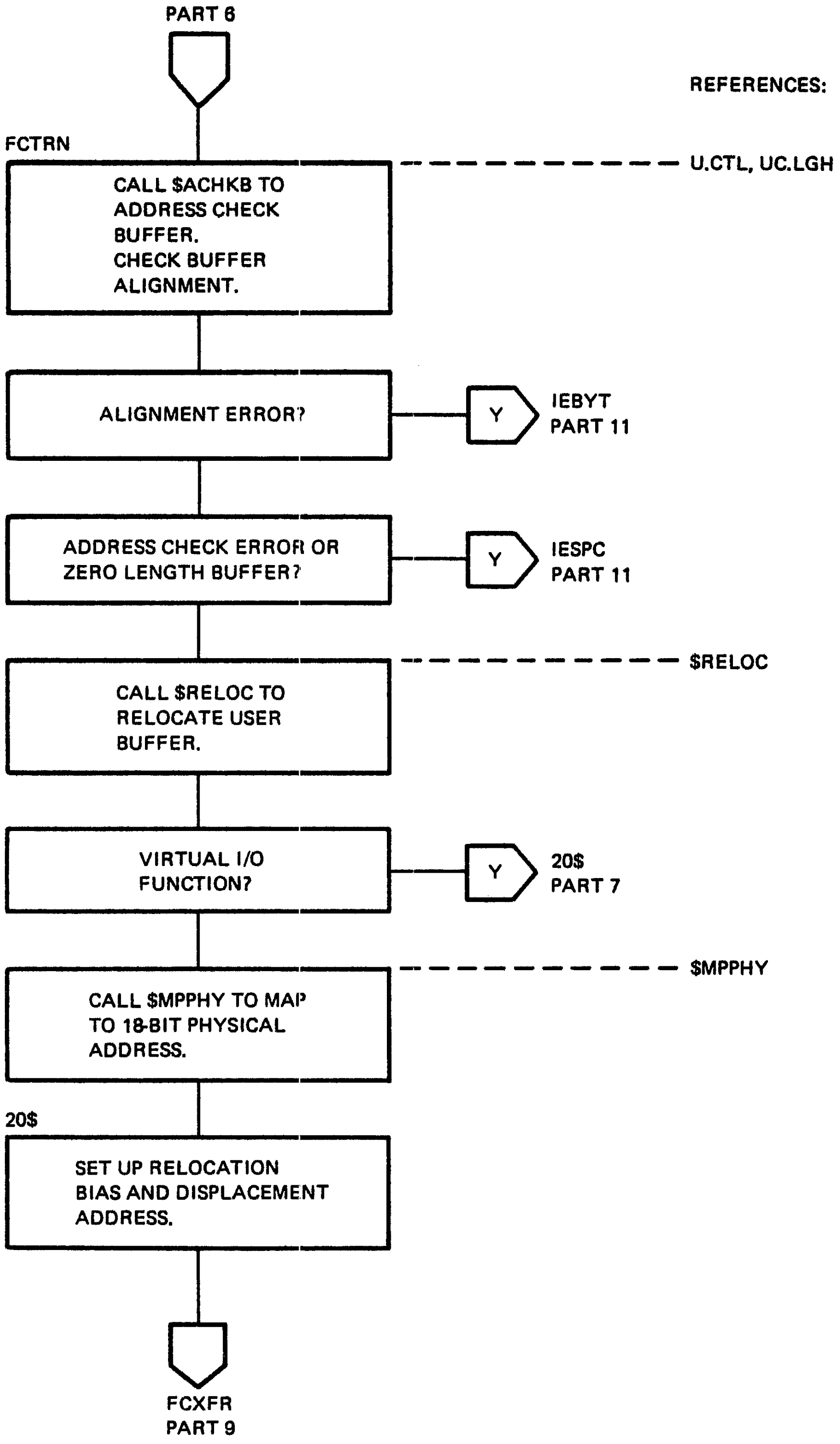

PART 6
REFERENCES:
FCTRN
CALL $ACHKB TO ADDRESS CHECK BUFFER. CHECK BUFFER ALIGNMENT.
U.CTL, UC.LGH
ALIGNMENT ERROR?
Y
IEBYT PART 11
ADDRESS CHECK ERROR OR ZERO LENGTH BUFFER?
Y
IESPC PART 11
CALL $RELOC TO RELOCATE USER BUFFER.
$RELOC
VIRTUAL I/O FUNCTION?
Y
20$ PART 7
CALL $MPPHY TO MAP TO 18-BIT PHYSICAL ADDRESS.
$MPPHY
20$
SET UP RELOCATION BIAS AND DISPLACEMENT ADDRESS.
FCXFR PART 9

Figure 6-2 QIO Directive Processing (Part 7 of 24)

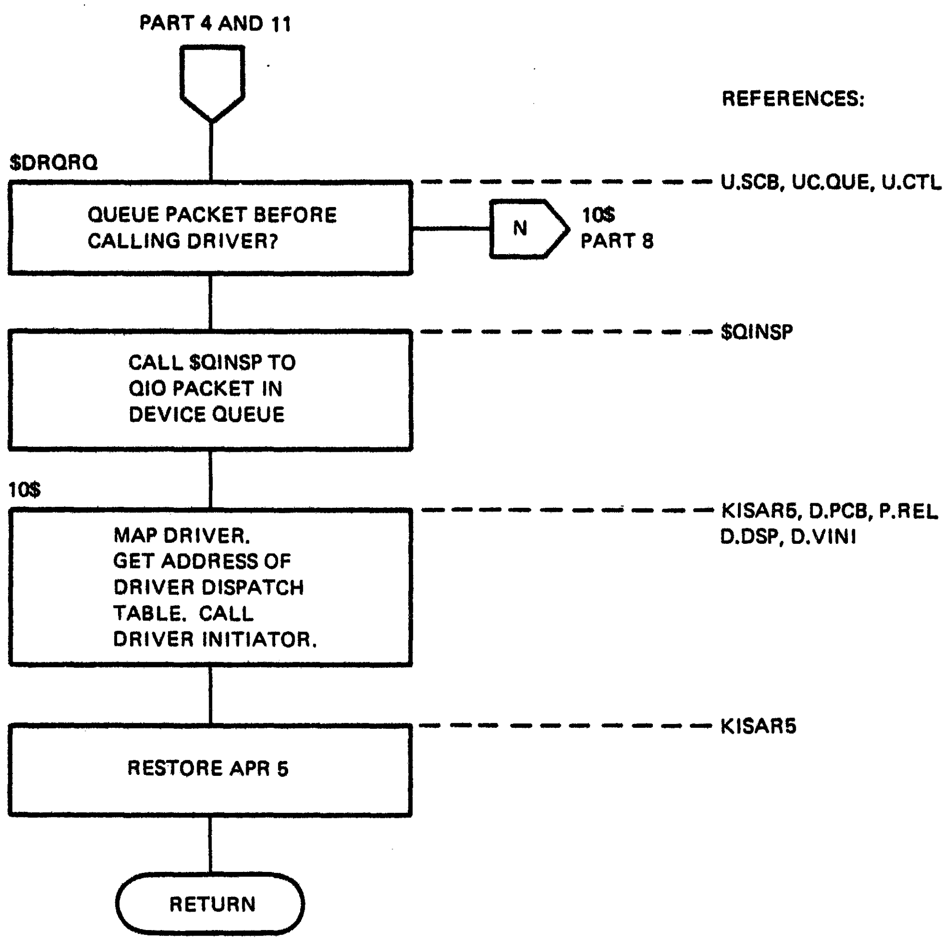

PART 4 AND 11
REFERENCES:
$DRQRQ
QUEUE PACKET BEFORE CALLING DRIVER?
U.SCB, UC.QUE, U.CTL
N
10$
PART 8
CALL $QINSP TO QIO PACKET IN DEVICE QUEUE
$QINSP
10$
MAP DRIVER. GET ADDRESS OF DRIVER DISPATCH TABLE. CALL DRIVER INITIATOR.
KISAR6, D.PCB, P.REL
D.DSP, D.VINI
RESTORE APR 5
KISAR5
RETURN

Figure 6-2 QIO Directive Processing (Part 8 of 24)

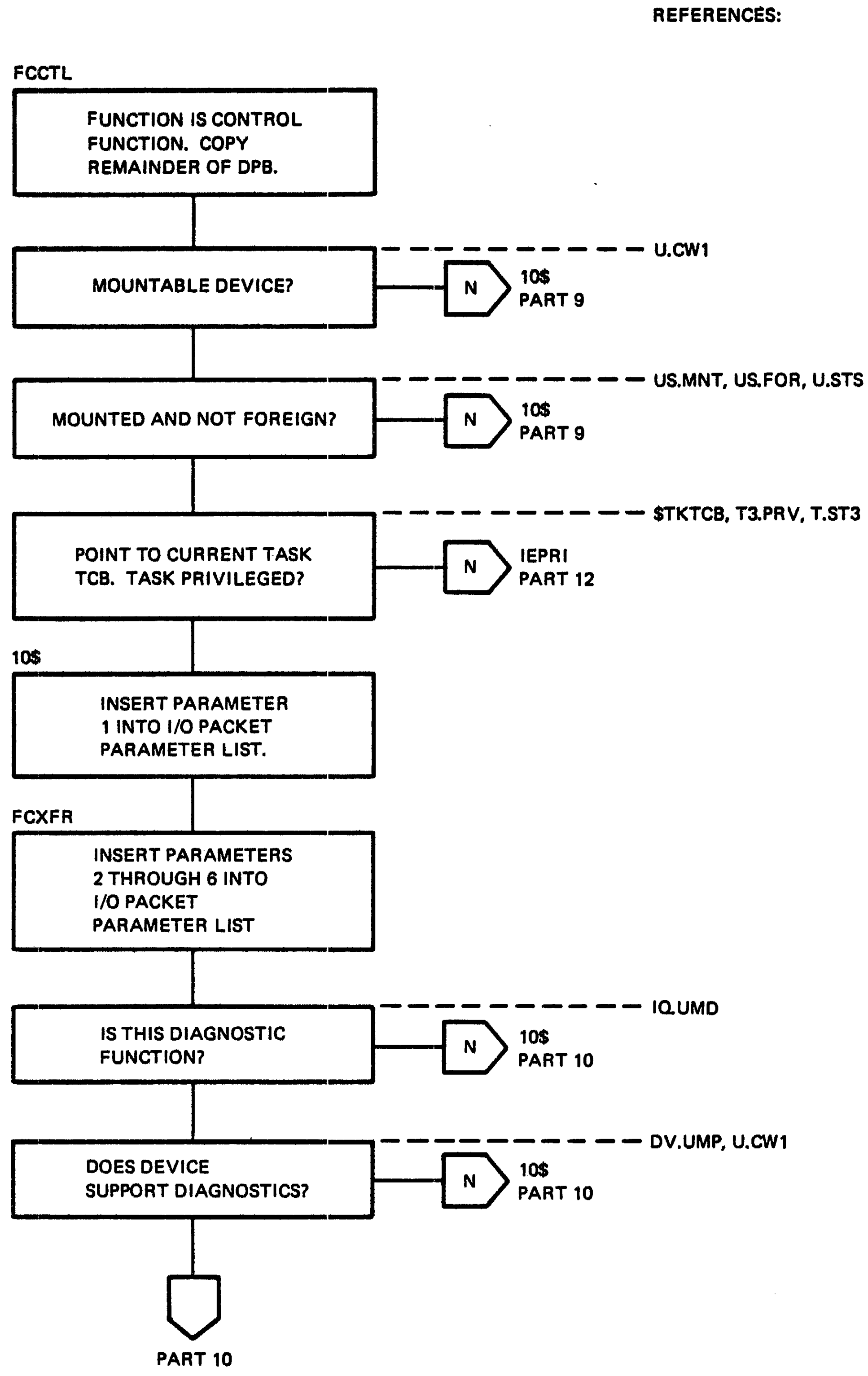

REFERENCES:
FCCTL
FUNCTION IS CONTROL FUNCTION. COPY REMAINDER OF DPB.
U.CW1
MOUNTABLE DEVICE?
N
10$
PART 9
US.MNT, US.FOR, U.STS
MOUNTED AND NOT FOREIGN?
N
10$
PART 9
$TKTCB, T3.PRV, T.ST3
POINT TO CURRENT TASK TCB. TASK PRIVILEGED?
N
IEPRI
PART 12
10$
INSERT PARAMETER 1 INTO I/O PACKET PARAMETER LIST.
FCXFR
INSERT PARAMETERS 2 THROUGH 6 INTO I/O PACKET PARAMETER LIST
IQ.UMD
IS THIS DIAGNOSTIC FUNCTION?
N
10$
PART 10
DV.UMP, U.CW1
DOES DEVICE SUPPORT DIAGNOSTICS?
N
10$
PART 10
PART 10

Figure 6-2 QIO Directive Processing (Part 9 of 24)

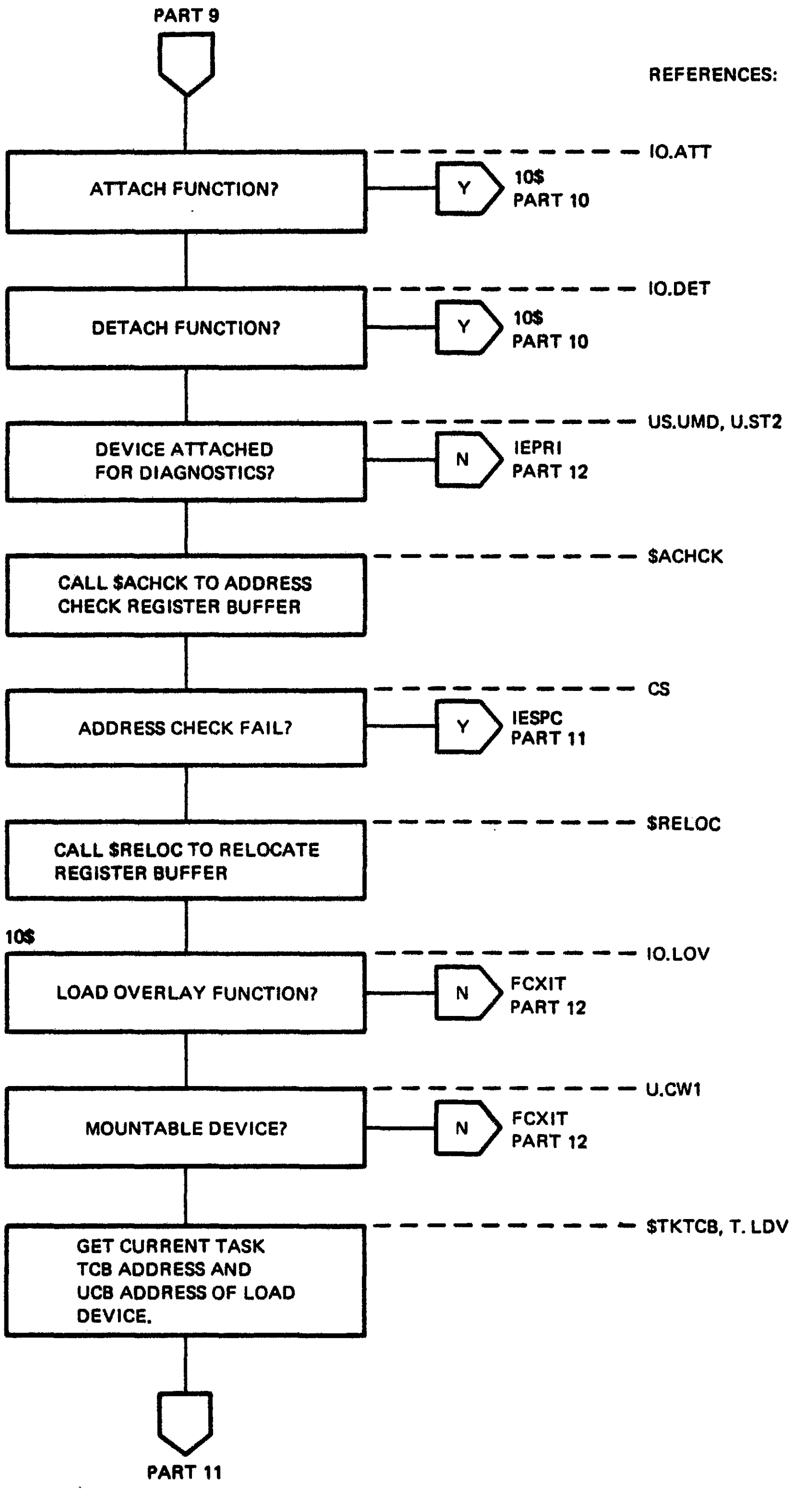


Figure 6-2 QIO Directive Processing (Part 10 of 24)

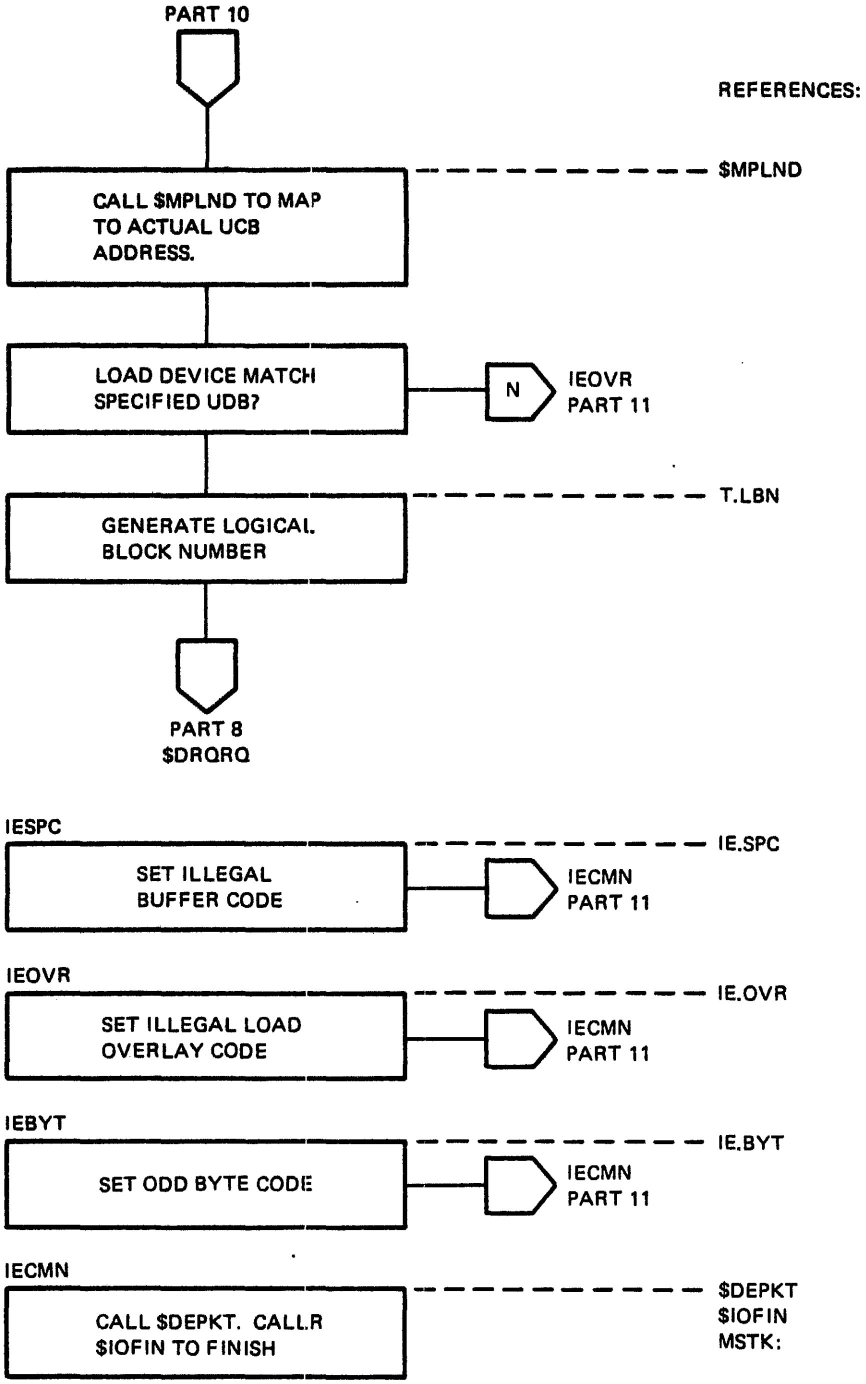


Figure 6-2 QIO Directive Processing (Part 11 of 24)

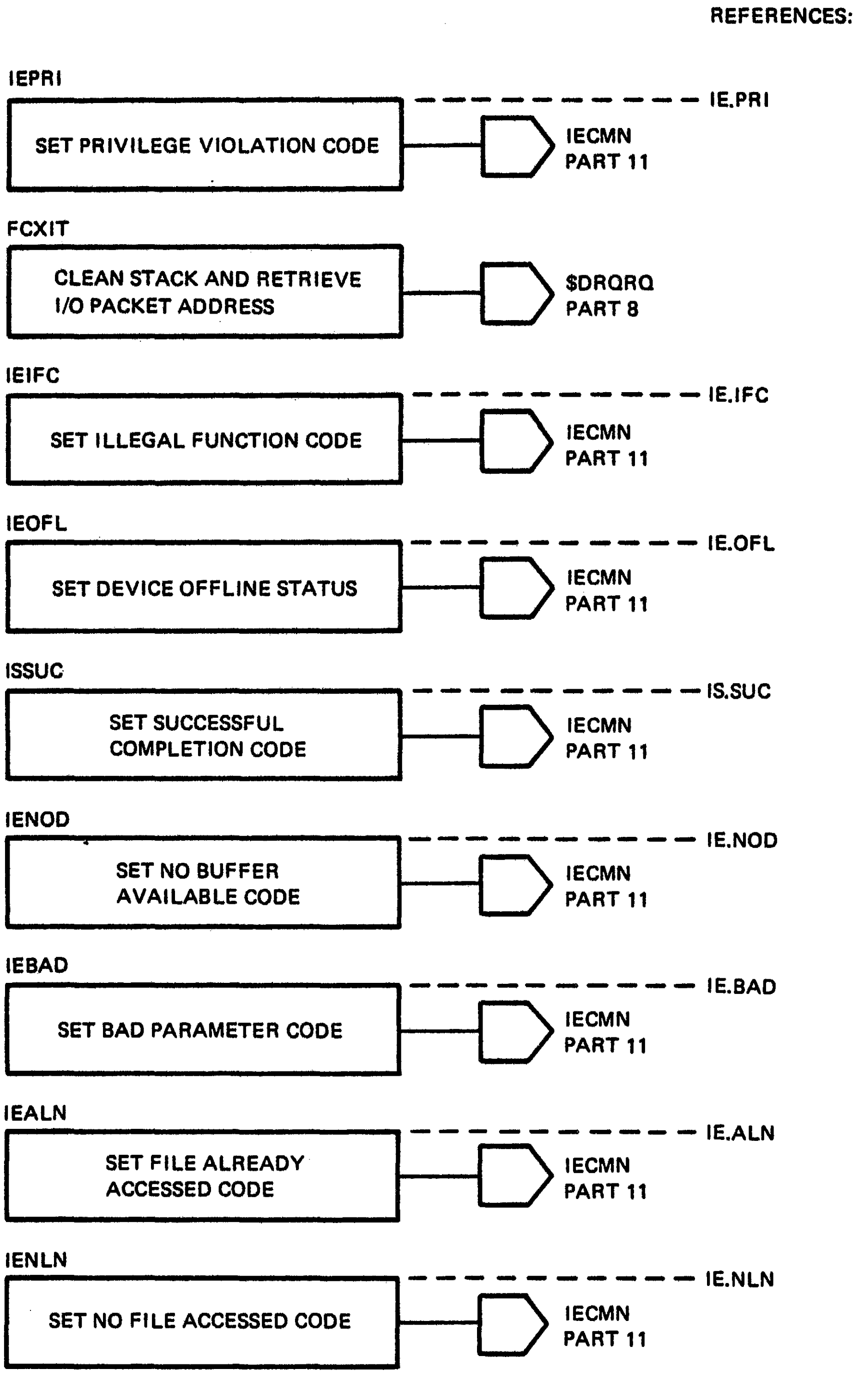


Figure 6-2 QIO Directive Processing (Part 12 of 24)

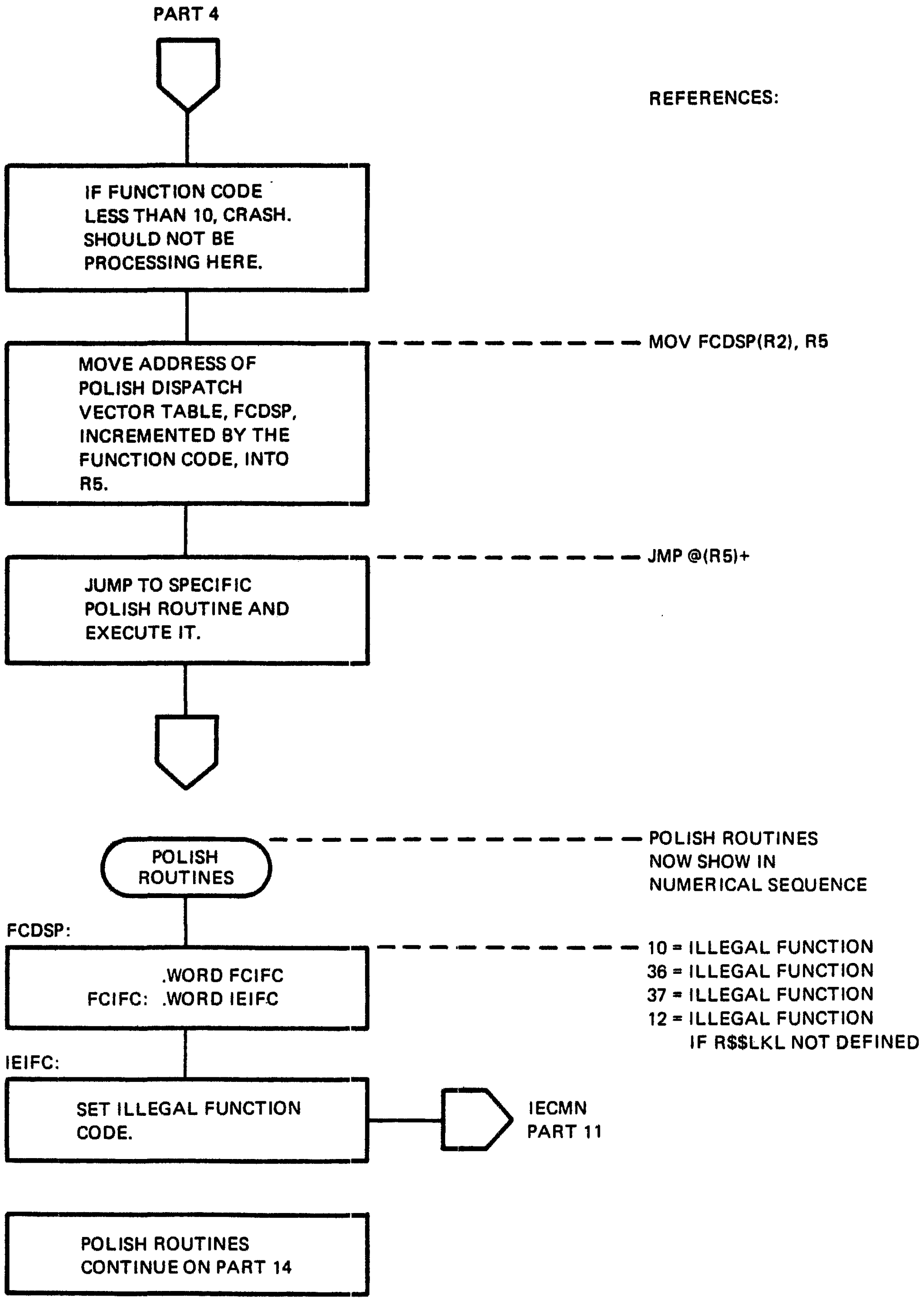

PART 4
REFERENCES:
IF FUNCTION CODE LESS THAN 10, CRASH. SHOULD NOT BE PROCESSING HERE.
MOVE ADDRESS OF POLISH DISPATCH VECTOR TABLE, FCDSP, INCREMENTED BY THE FUNCTION CODE, INTO R5.
MOV FCDSP(R2), R5
JUMP TO SPECIFIC POLISH ROUTINE AND EXECUTE IT.
JMP @(R5)+
POLISH ROUTINES
POLISH ROUTINES NOW SHOW IN NUMERICAL SEQUENCE
FCDSP:
.WORD FCIFC
FCIFC: .WORD IEIFC
10 = ILLEGAL FUNCTION
36 = ILLEGAL FUNCTION
37 = ILLEGAL FUNCTION
12 = ILLEGAL FUNCTION
IF R$$LKL NOT DEFINED
IEIFC:
SET ILLEGAL FUNCTION CODE.
IECMN PART 11
POLISH ROUTINES CONTINUE ON PART 14

Figure 6-2 QIO Directive Processing (Part 13 of 24)

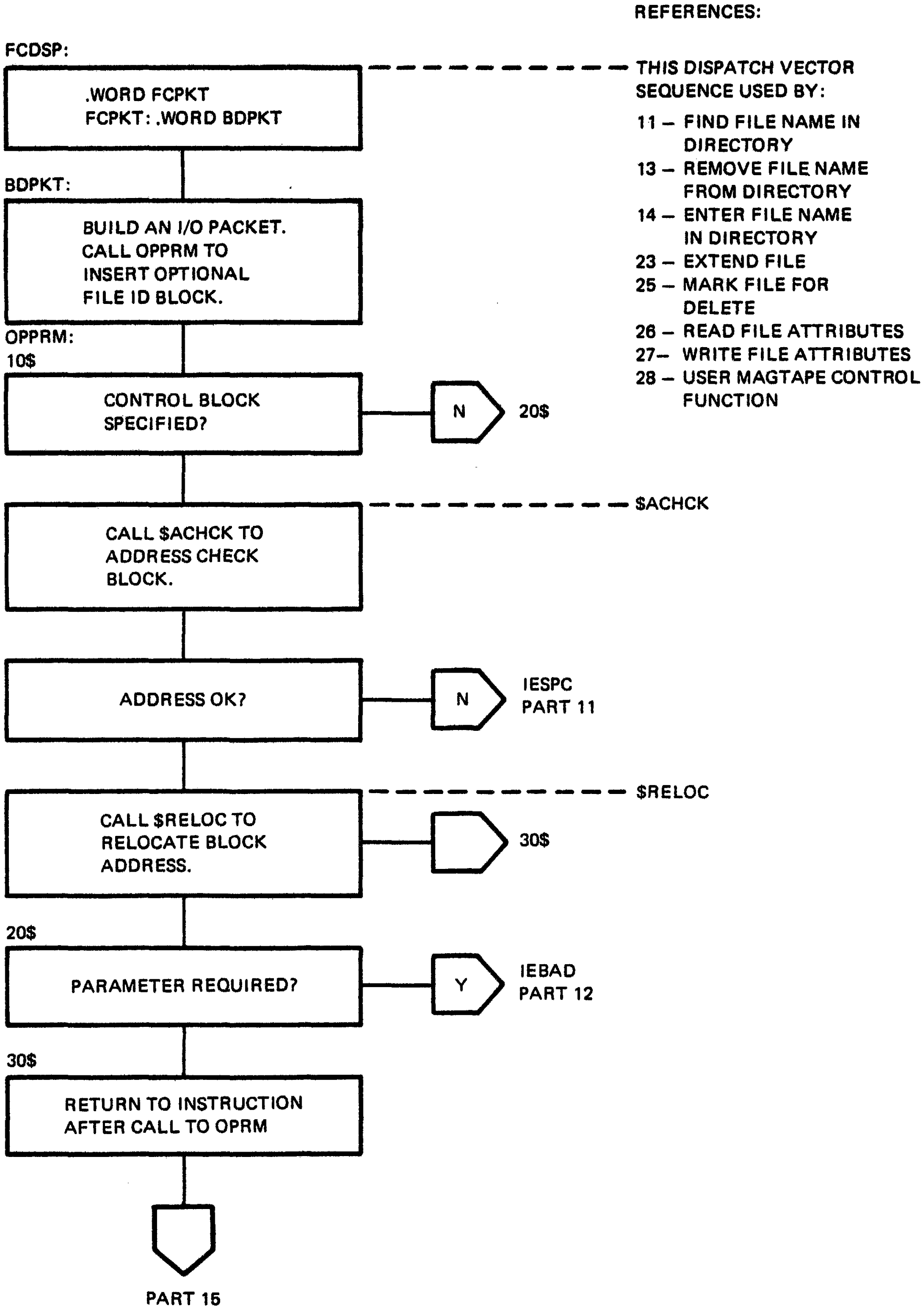

REFERENCES:
FCDSP:
.WORD FCPKT
FCPKT: .WORD BDPKT
THIS DISPATCH VECTOR SEQUENCE USED BY:
11 – FIND FILE NAME IN DIRECTORY
13 – REMOVE FILE NAME FROM DIRECTORY
14 – ENTER FILE NAME IN DIRECTORY
23 – EXTEND FILE
25 – MARK FILE FOR DELETE
26 – READ FILE ATTRIBUTES
27– WRITE FILE ATTRIBUTES
28 – USER MAGTAPE CONTROL FUNCTION
BDPKT:
BUILD AN I/O PACKET. CALL OPPRM TO INSERT OPTIONAL FILE ID BLOCK.
OPPRM:
10$
CONTROL BLOCK SPECIFIED?
N
20$
$ACHCK
CALL $ACHCK TO ADDRESS CHECK BLOCK.
ADDRESS OK?
N
IESPC
PART 11
$RELOC
CALL $RELOC TO RELOCATE BLOCK ADDRESS.
30$
20$
PARAMETER REQUIRED?
Y
IEBAD
PART 12
30$
RETURN TO INSTRUCTION AFTER CALL TO OPRM
PART 15

Figure 6-2 QIO Directive Processing (Part 14 of 24)

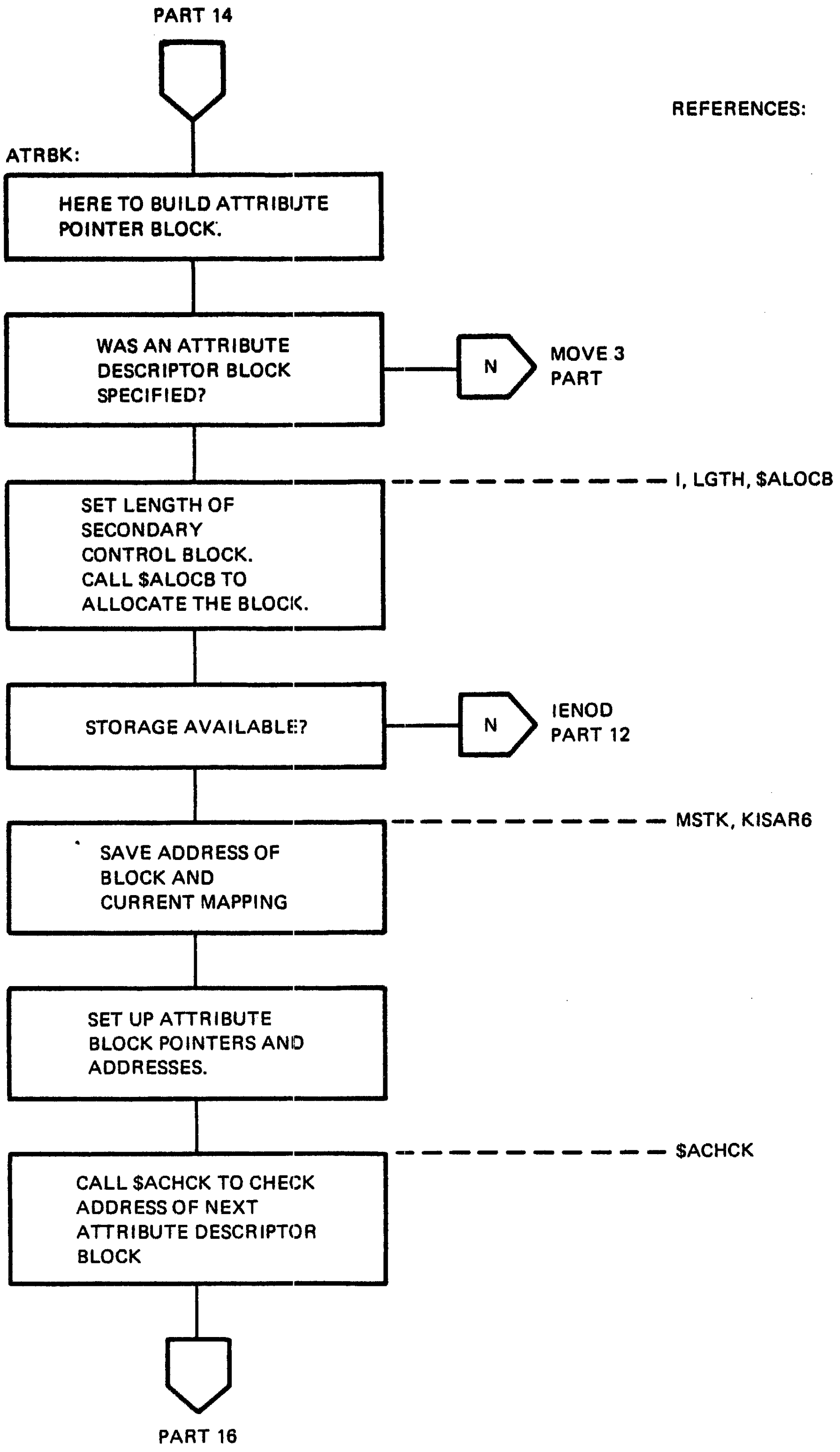

PART 14
REFERENCES:
ATRBK:
HERE TO BUILD ATTRIBUTE POINTER BLOCK.
WAS AN ATTRIBUTE DESCRIPTOR BLOCK SPECIFIED?
N
MOVE 3 PART
I, LGTH, $ALOCB
SET LENGTH OF SECONDARY CONTROL BLOCK. CALL $ALOCB TO ALLOCATE THE BLOCK.
STORAGE AVAILABLE?
N
IENOD PART 12
MSTK, KISAR6
SAVE ADDRESS OF BLOCK AND CURRENT MAPPING
SET UP ATTRIBUTE BLOCK POINTERS AND ADDRESSES.
$ACHCK
CALL $ACHCK TO CHECK ADDRESS OF NEXT ATTRIBUTE DESCRIPTOR BLOCK
PART 16

Figure 6-2 QIO Directive Processing (Part 15 of 24)

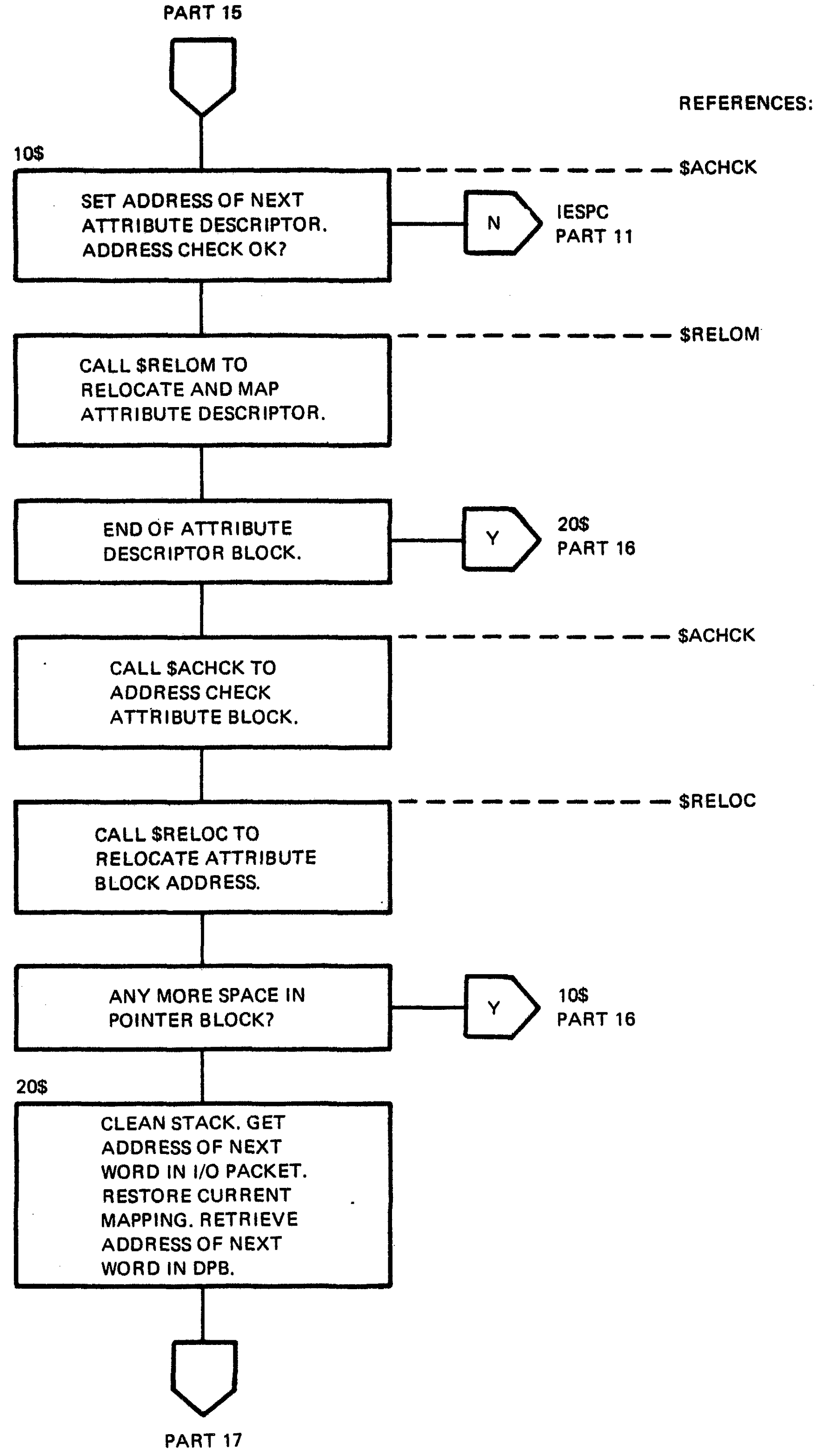

PART 15
REFERENCES:
10$
SET ADDRESS OF NEXT ATTRIBUTE DESCRIPTOR. ADDRESS CHECK OK?
$ACHCK
N
IESPC PART 11
CALL $RELOM TO RELOCATE AND MAP ATTRIBUTE DESCRIPTOR.
$RELOM
END OF ATTRIBUTE DESCRIPTOR BLOCK.
Y
20$ PART 16
CALL $ACHCK TO ADDRESS CHECK ATTRIBUTE BLOCK.
$ACHCK
CALL $RELOC TO RELOCATE ATTRIBUTE BLOCK ADDRESS.
$RELOC
ANY MORE SPACE IN POINTER BLOCK?
Y
10$ PART 16
20$
CLEAN STACK. GET ADDRESS OF NEXT WORD IN I/O PACKET. RESTORE CURRENT MAPPING. RETRIEVE ADDRESS OF NEXT WORD IN DPB.
PART 17

Figure 6-2 QIO Directive Processing (Part 16 of 24)

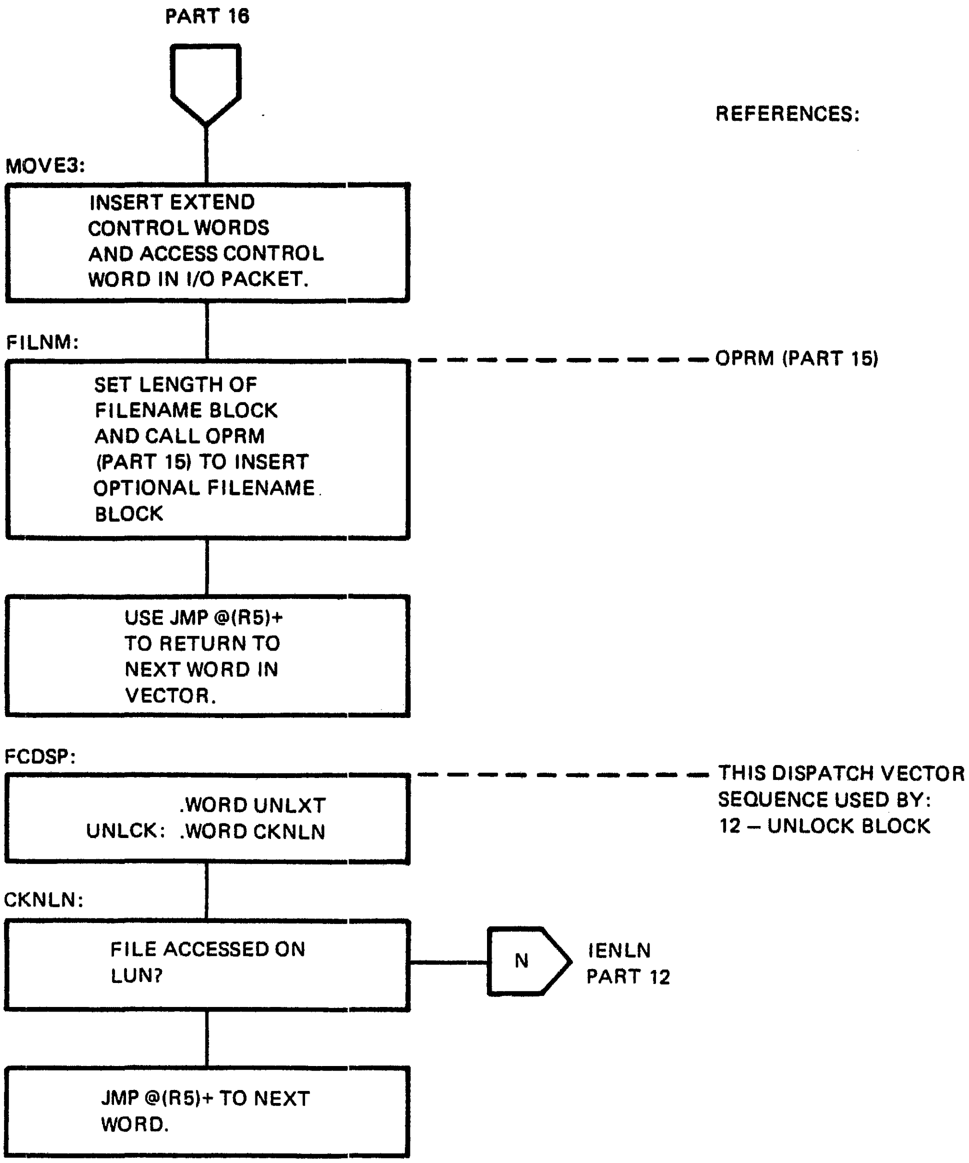

PART 16
REFERENCES:
MOVE3:
INSERT EXTEND CONTROL WORDS AND ACCESS CONTROL WORD IN I/O PACKET.
FILNM:
SET LENGTH OF FILENAME BLOCK AND CALL OPRM (PART 15) TO INSERT OPTIONAL FILENAME BLOCK
OPRM (PART 15)
USE JMP @(R5)+ TO RETURN TO NEXT WORD IN VECTOR.
FCDSP:
.WORD UNLXT
UNLCK: .WORD CKNLN
THIS DISPATCH VECTOR SEQUENCE USED BY:
12 – UNLOCK BLOCK
CKNLN:
FILE ACCESSED ON LUN?
N
IENLN
PART 12
JMP @(R5)+ TO NEXT WORD.

Figure 6-2 QIO Directive Processing (Part 17 of 24)

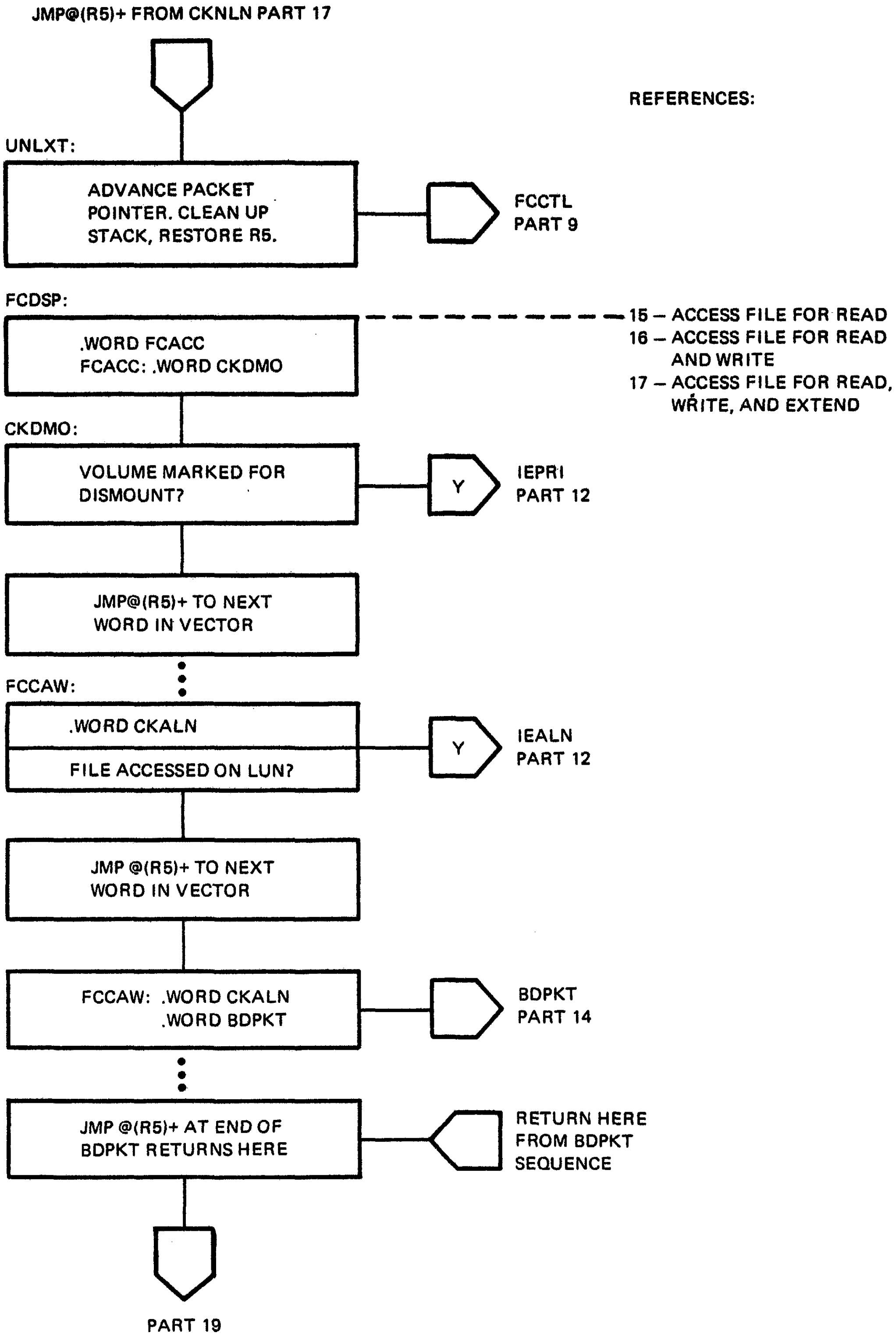

JMP@(R5)+ FROM CKNLN PART 17
REFERENCES:
UNLXT:
ADVANCE PACKET POINTER. CLEAN UP STACK, RESTORE R5.
FCCTL PART 9
FCDSP:
.WORD FCACC
FCACC: .WORD CKDMO
15 – ACCESS FILE FOR READ
16 – ACCESS FILE FOR READ AND WRITE
17 – ACCESS FILE FOR READ, WRITE, AND EXTEND
CKDMO:
VOLUME MARKED FOR DISMOUNT?
Y
IEPRI PART 12
JMP@(R5)+ TO NEXT WORD IN VECTOR
FCCAW:
.WORD CKALN
FILE ACCESSED ON LUN?
Y
IEALN PART 12
JMP @(R5)+ TO NEXT WORD IN VECTOR
FCCAW: .WORD CKALN
.WORD BDPKT
BDPKT PART 14
JMP @(R5)+ AT END OF BDPKT RETURNS HERE
RETURN HERE FROM BDPKT SEQUENCE
PART 19

Figure 6-2 QIO Directive Processing (Part 18 of 24)

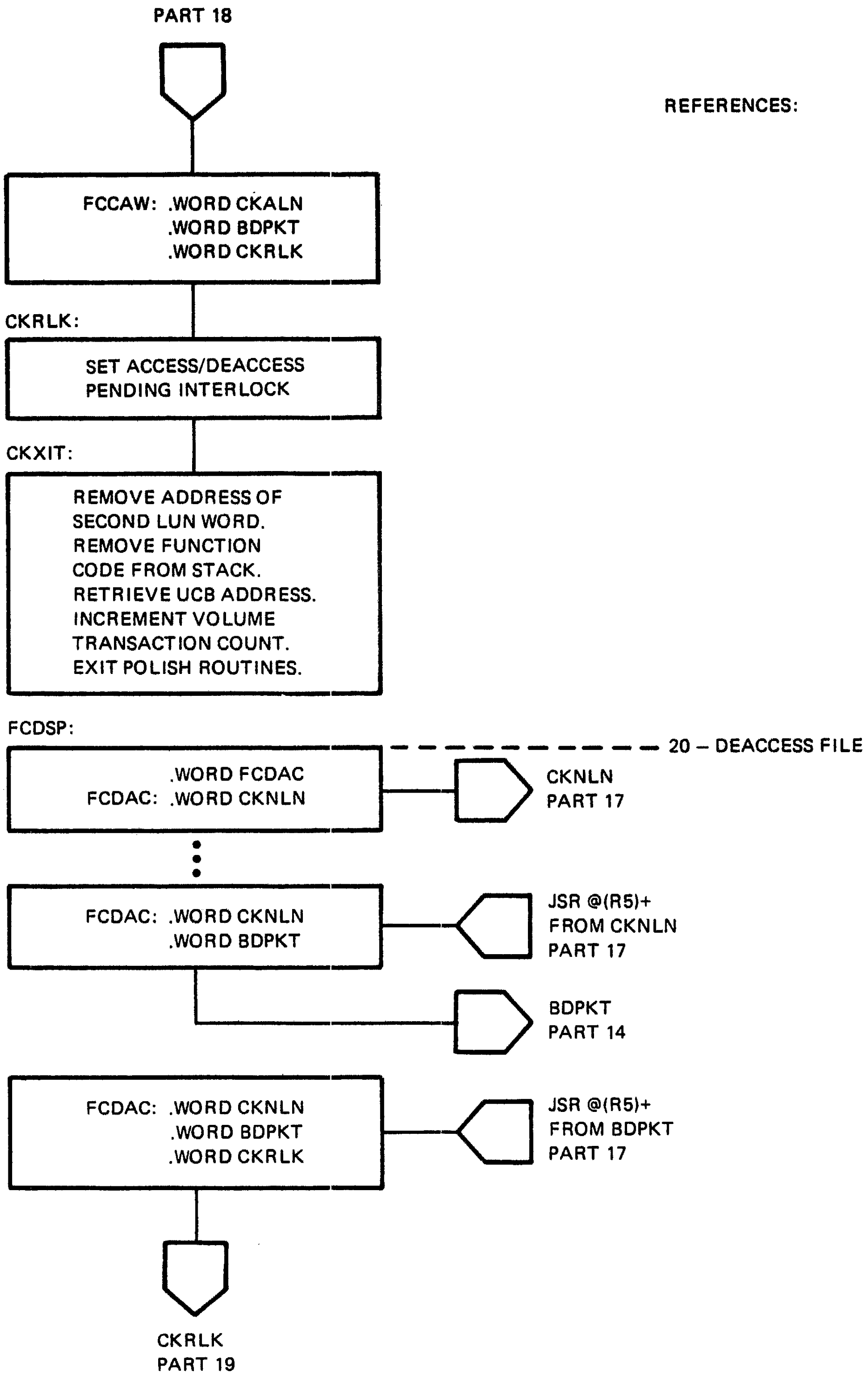


Figure 6-2 QIO Directive Processing (Part 19 of 24)

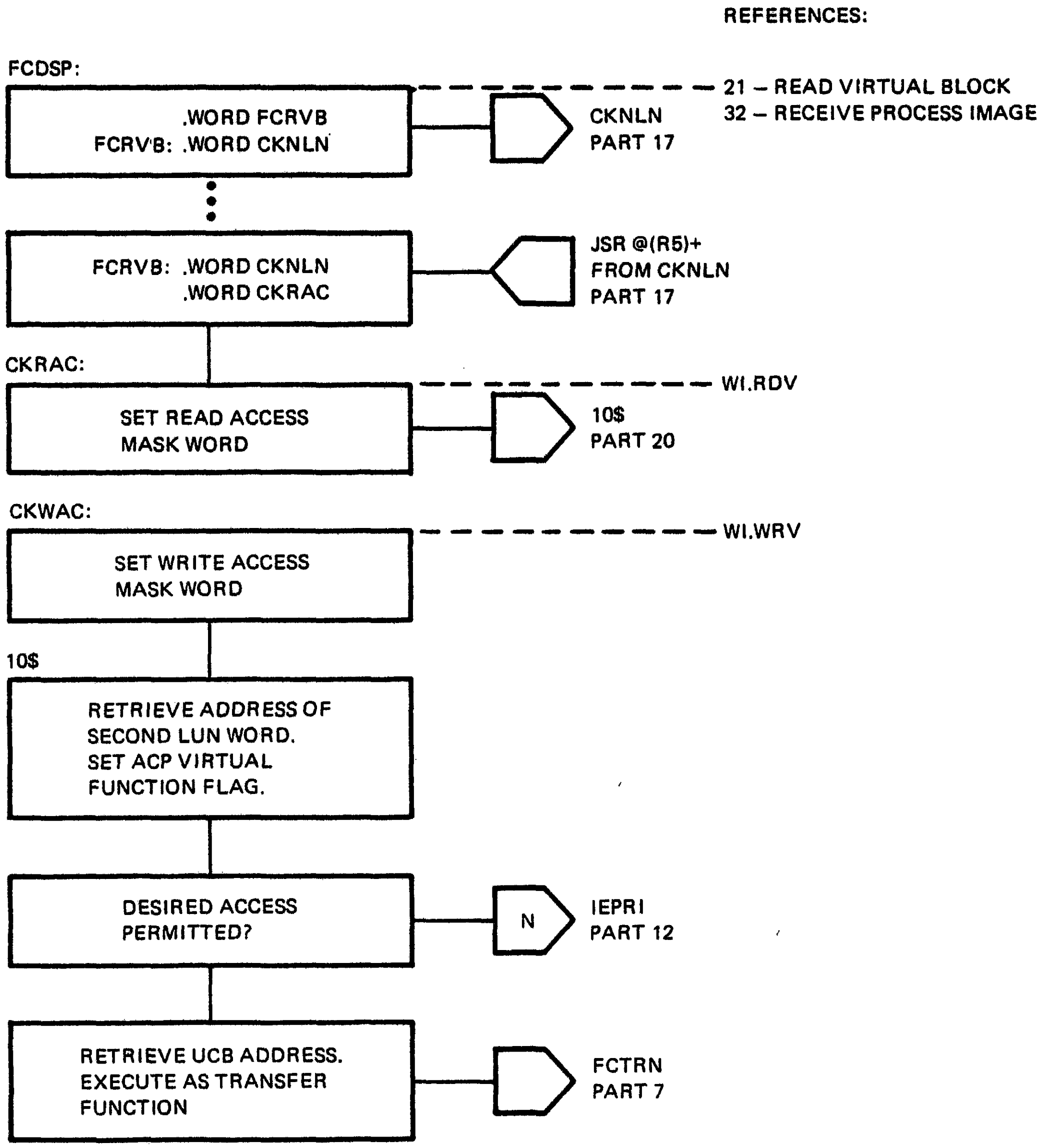


Figure 6-2 QIO Directive Processing (Part 20 of 24)

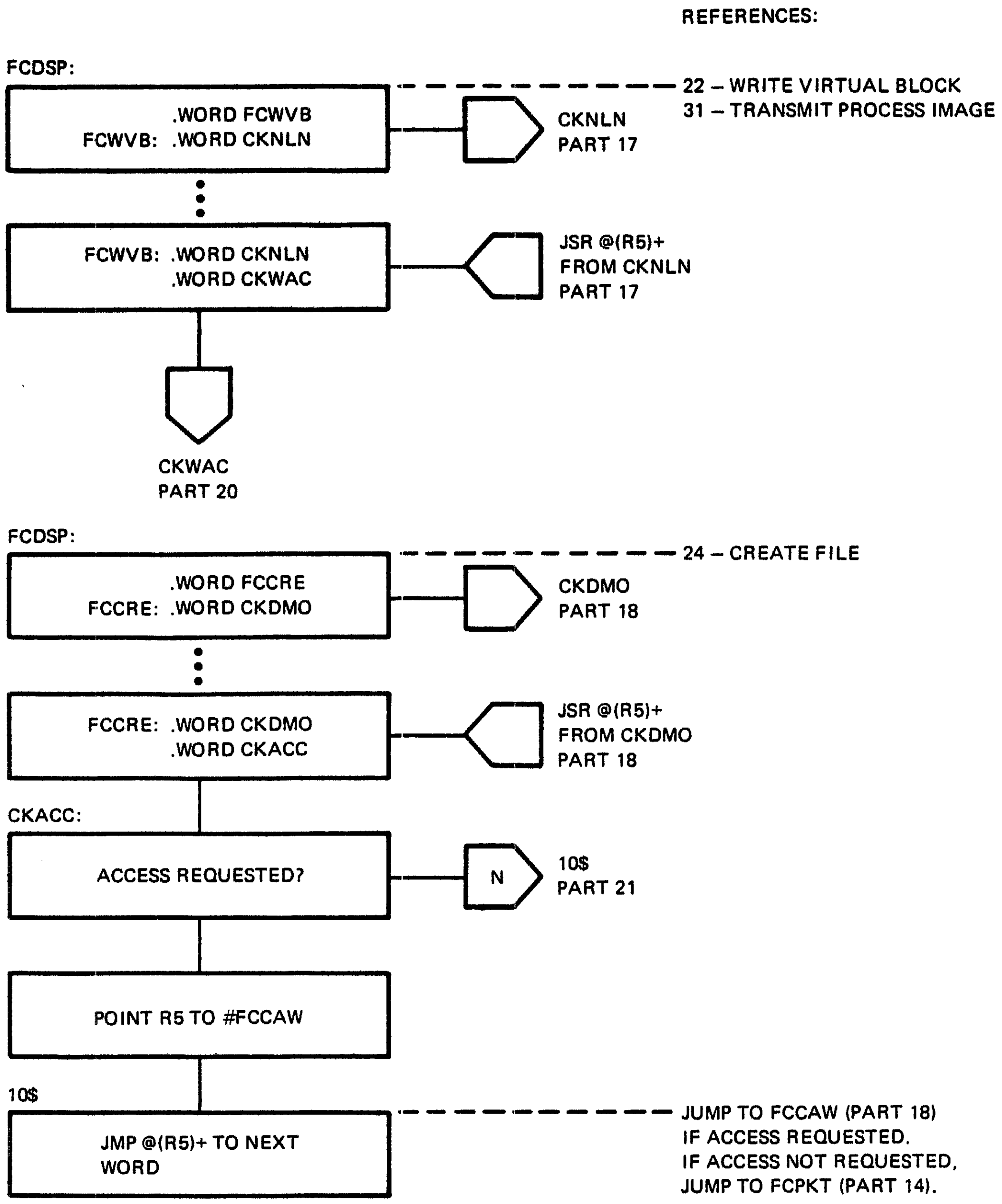


Figure 6-2 QIO Directive Processing (Part 21 of 24)

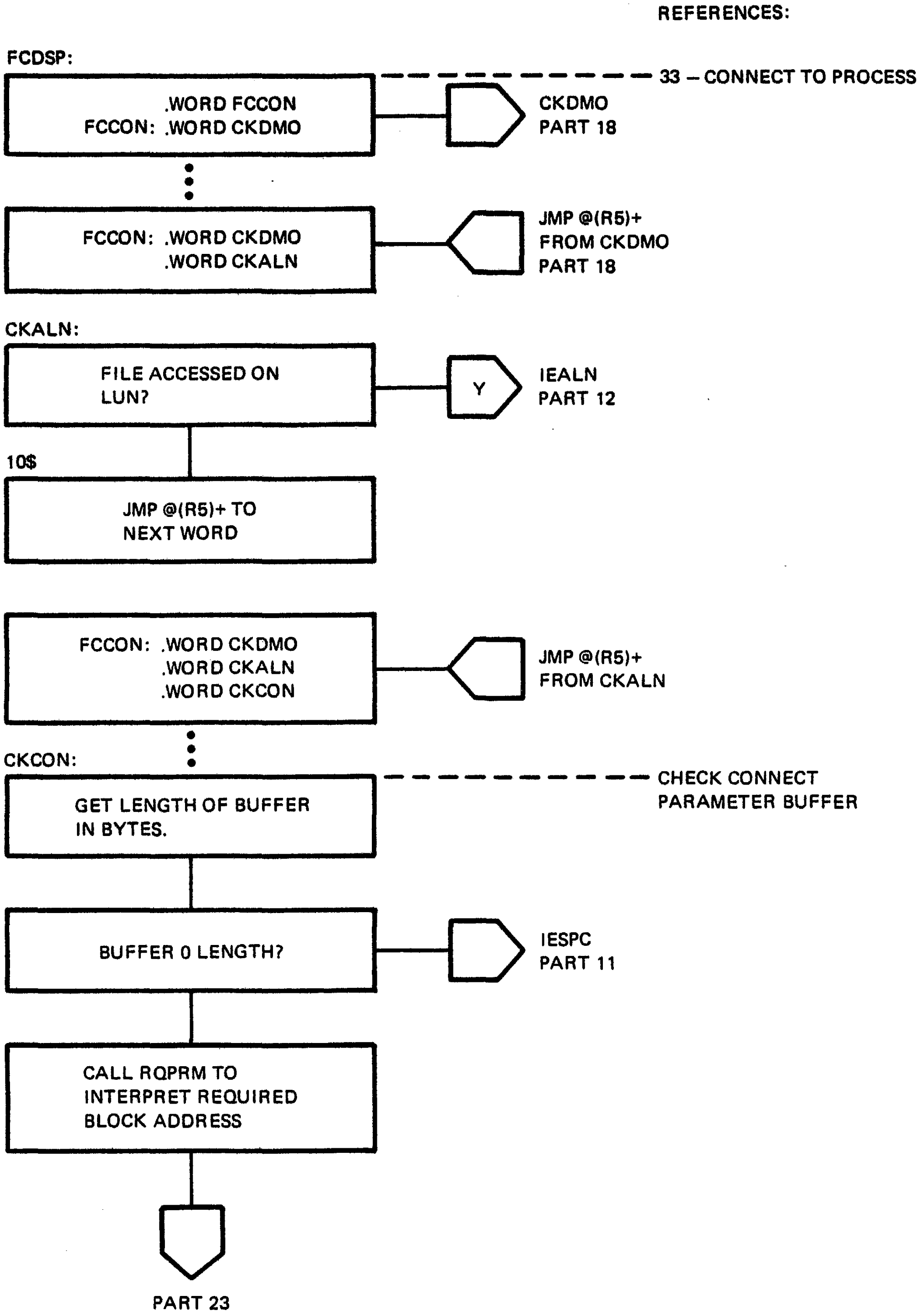


Figure 6-2 QIO Directive Processing (Part 22 of 24)

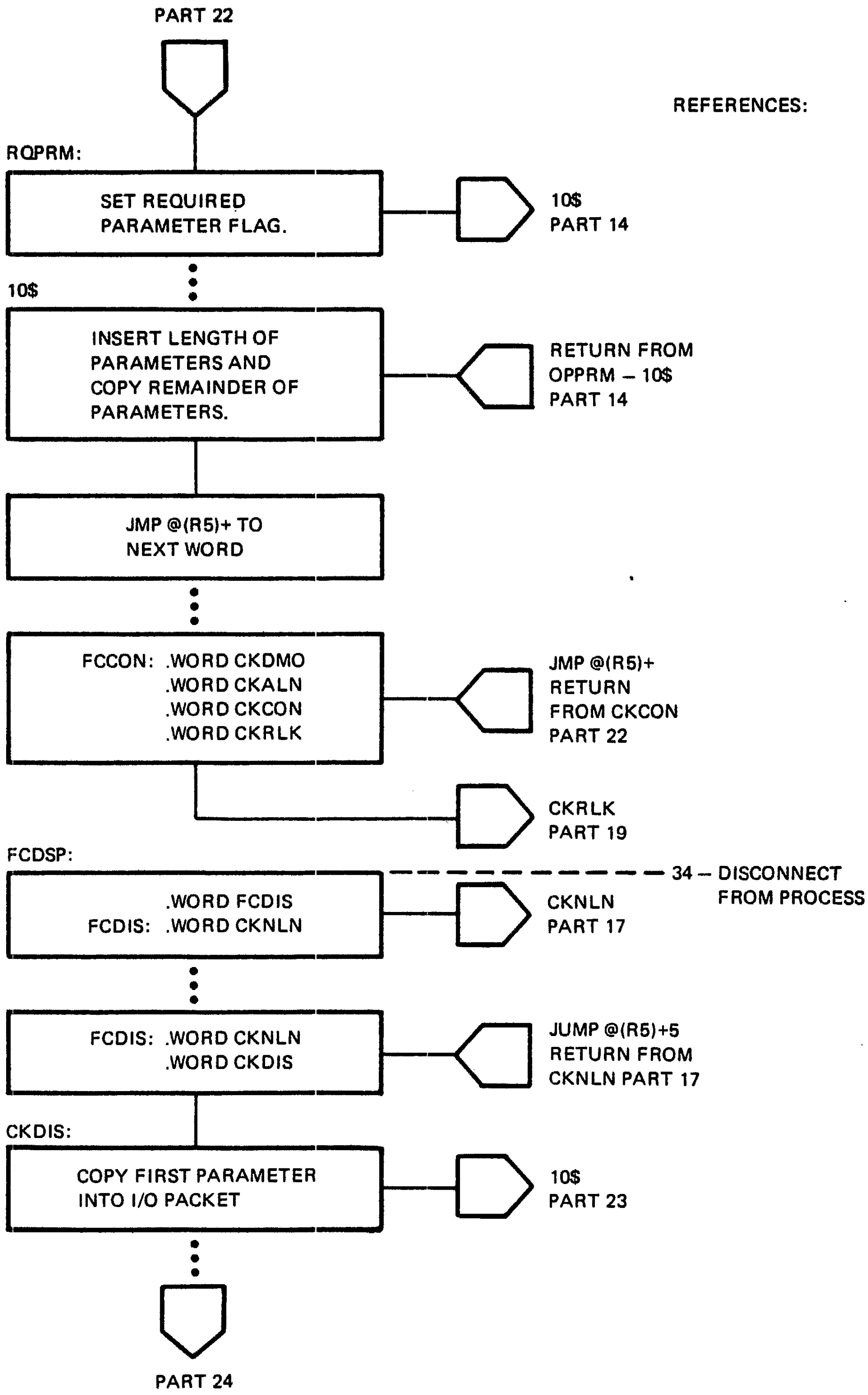


Figure 6-2 QIO Directive Processing (Part 23 of 24)

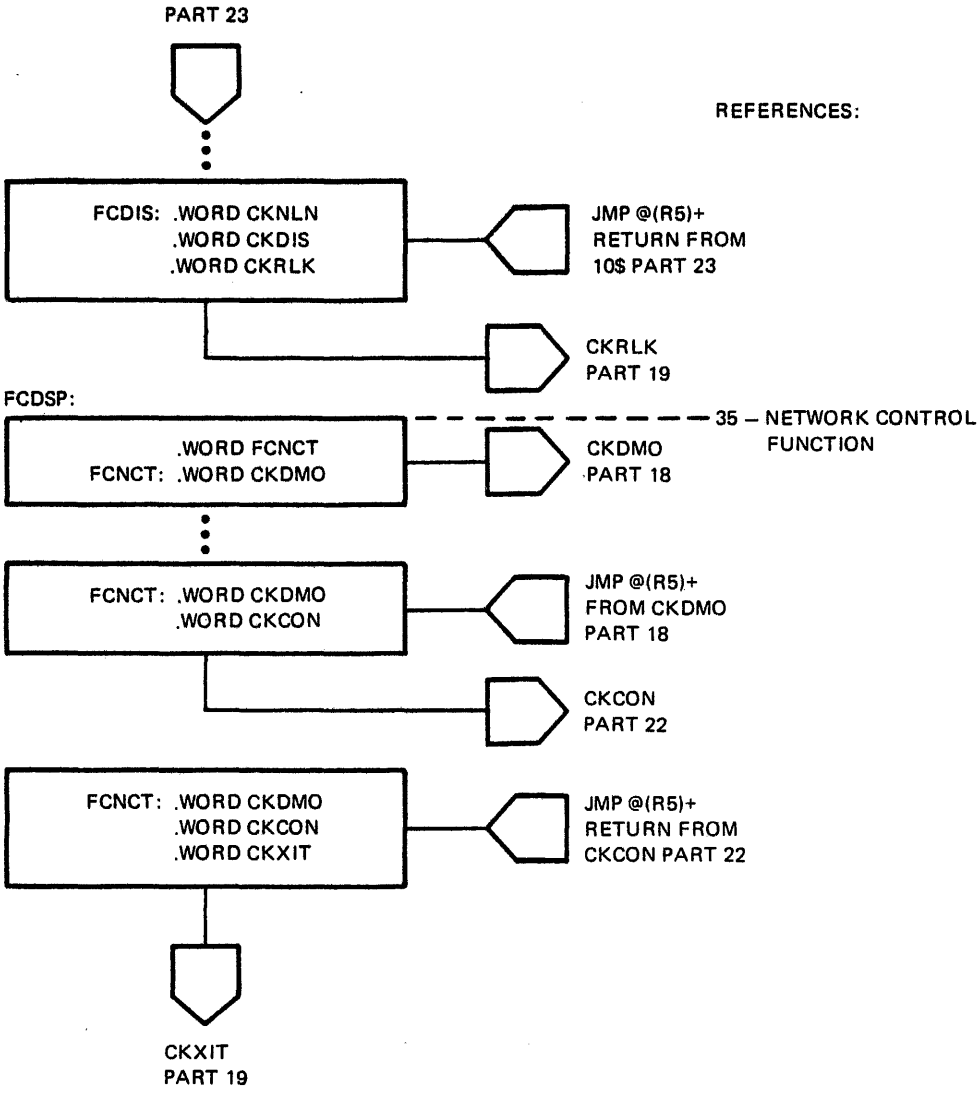


Figure 6-2 QIO Directive Processing (Part 24 of 24)

CHAPTER 7

MODULE DESCRIPTIONS

This chapter describes the modules, as individual units, that perform the RSX-11M Executive programming functions.

The modules are described in alphabetical order and the description of a new module always begins on a new page.

7.1 CHAPTER ORGANIZATION

The following information is presented in this chapter:

- Module name
- Macro Library Calls
- Entry points (routines)
- Function of this module or routines
- Modules this module calls
- Entry (input) conditions
- Exit (output) conditions

The modules appear alphabetically and are grouped by system component. Thus, this chapter describes the Executive modules, then the FCP modules, and finally, the MCR modules; all alphabetically.

7.1.1 Module Name

The module name starts the description of each module. The name always appears at the top of a new page following the description of the previous module. A brief statement of the function of the module is included here.

7.1.2 Macro Library Calls

These calls appear in the code listing after the copyright notice. They are included here to show the data areas or control blocks to which the module refers.

7.1.3 Entry Points

The entry point is the global label in the module to which program control is transfered by another part of the program. It can also be considered as the label of a routine within the module. Statements about the function of the routine are included here.

7.1.4 Calls

Included here are all the calls (using the CALL or CALLR macro) that this routine executes. They are listed in order of appearance in the code. However, they may appear more than once in the code.

7.1.5 Entry (input) Conditions

Entry conditions are those conditions set up by the routine that called this routine or are present when this routine starts processing. These parameters or conditions are in data areas or registers in the form of data or addresses. The contents of these registers or data areas may be altered by this routine.

7.1.6 Exit (output) Conditions

Exit conditions may be changed register contents, data area contents, stack contents, or status indicators. Usually, the output is needed by some other part of the system program. However, the routine or module may perform a function without producing anything that can be defined easily as "output". In many of these cases, the routine indicates that it completed its operation by altering the C-bit or returning a directive status (for directive processing routines).

7.2 EXECUTIVE MODULE DESCRIPTIONS

7.2.1 BFCTL Module

BFCTL — The BFCTL module puts a byte or word into the user's buffer or gets a byte or a word out of the user's buffer. Also, moves data one byte at a time from place to place in the memory of a mapped system.

Macro Library Calls
HWDDF$ — Define hardware registers

Entry Point -
$GTBYT:: — The GTBYT routine maps to the user buffer if Memory Management is defined. GTBYT gets the next byte from the user buffer and returns it to the caller on the stack. After the byte has been fetched, $GTBYT increments the next byte address.

Calls — None

Input — R5 = Address of the UCB that contains the buffer pointers.

Output — The stack contains the byte from the user buffer.

Note — All registers are preserved across the call.

Entry Point -
$PTBYT:: — The PTBYT routine maps to the user buffer if Memory Management is defined. PTBYT puts a byte in the next location in the user buffer and increments the next byte address.

Calls — None

Input — R5 = Address of the UCB that contains the buffer pointers.

Stack = Byte to be stored in the next location in the user buffer

Output — $PTBYT removes the byte from the stack and stores it in the user buffer. $PTBYT increments the next byte address.

Note — All registers are preserved across the call.

Entry Point -
$GTWRD:: — Gets the next word from the user buffer and returns it to the caller on the stack. After the word has been fetched, $GTWRD calculates the next word address.

Calls — None

Input — R5 = Address of the UCB that contains the buffer pointers.

Output — $GTWRD fetches the next word from the user buffer and returns it to the caller on the stack.

Stack = Next word of user buffer

Note — All registers are preserved across the call.

Entry Point -
$PTWRD:: — The PTWRD routine puts a word in the next location in the user buffer. After storing the word, $PTWRD calculates the next word address.

Calls — None

Input — R5 = the address of the UCB that contains the buffer pointers.

Stack = Word to be stored in the next location of the buffer.

Output — $PTWRD removes the word from the stack and stores it in the user buffer.

Note — All registers are preserved across the call.

Entry Point -
$GTCWD:: The GTCWD routine gets the next word from the user control buffer and returns it to the caller on the stack.

Calls None

Input R5 = Address of the UCB that contains the buffer pointers.

Output $GTCWD fetches the next word from the user control buffer and returns it to the caller on the stack.
Stack = next word of user buffer

Note All registers are preserved across the call.

Entry Point -
$BLXIO:: The BLXIO routine moves data within the memory of a mapped system.

Calls None

Input R0 = Number of bytes to move
R1 = Source APR5 bias
R2 = Source displacement
R3 = Destination APR6 bias
R4 = Destination displacement

Output After BLXIO finishes moving the data:
R0 and R5 are altered
R1 and R3 are preserved
R2 points to the last byte of the source +1
R4 points to the last byte of the destination +1

7.2.2 CORAL Module

CORAL This module contains the core allocation routines. The memory search algorithm looks for the first block of memory that is available for allocation. The size is rounded upward to a multiple of four bytes.

Macro Library Calls
CLKDF$ Define clock queue control block offsets
PKTDF$ Define I/O packet offsets

Entry Point -
$ALOCB:: The ALOCB routine allocates a core buffer
$ALOC1:: The ALOC1 routine allocates a core buffer (alternate entry)

Calls None

Input R0 = Address of core allocation listhead -2 if entry is at $ALOC1.
R1 = Size of the core buffer to allocate in bytes

Output C = 1 if insufficient core is available to allocate the block
C = 0 if the block is allocated
R0 = Address of the allocated block
R1 = Length of the allocated block

Entry Point -

$ALCLK:: The ALCLK routine allocates a core block for a clock queue entry. This routine stores the length of the clock block in R1.

Calls None

Input None

Output If core is not available to allocate the block, $ALCLK returns a 'D.RS1' directive status. If enough core is available, $ALCLK returns the address of the allocated block to the caller in R0.

Note The ALCLK routine shares common code with the ALPKT routine.

Entry Point -

$ALPKT:: The ALPKT routine allocates a core block for a SEND or I/O REQUEST queue entry. $ALPKT stores the length of the I/O Packet (I.LGTH) in R1 and calls $ALOCB to allocate the block.

Calls $ALOCB

Input None

Output If core is not available to allocate the block, $ALPKT returns a directive status of 'D.RS1'. Otherwise, $ALKPT returns the address of the allocated block to the caller in R0.

Entry Point -

$DEACB:: The DEACB routine deallocates the core buffer.

$DEAC1:: The DEAC1 routine deallocates the core buffer (alternate entry).

DEAC inserts the Executive core block to be deallocated into the free block chain by core address. If an adjacent block is currently free, DEAC merges the two blocks and inserts them in the free block chain.

Calls None

Input R0 = Address of the core buffer to be deallocated
R1 = Size of the core buffer to deallocate in bytes
R3 = Address of core allocation listhead -2 if entry is at $DEAC1

Output DEAC merges the core block into the free block chain by core address and agglomerates it with adjacent free blocks.

Error The system crashes if deallocation is attempted before the front or past the end of the system pool.

Entry Point -

$DECLK:: The DECLK routine deallocates a core block that was used for a clock queue entry.

Calls None (branches to $DEACB)

Input R0 = Address of the core block to be deallocated

Output DECLK deallocates the clock queue entry core block and agglomerates adjacent free core blocks.

Entry Point -
$DEPKT:: The DEPKT routine deallocates a core block that was used for a SEND or I/O REQUEST queue entry.

Calls None

Input R0 = Address of the core block to be deallocated.

Output The core block is deallocated.

Note This routine uses the code in the DEACB routine.

7.2.3 CRASH Module

CRASH This module is entered via a JUMP whenever a fatal system error is detected. This routine dumps memory on a DECtape, RK05, TU10 or TU16. The first block of the dump contains information about the state of the system at the time of the crash. When the dump is finished, you may reboot the system or re-execute the dump.

This module uses the following local data and routines:

```
CRSMSG:  Message; /CRASH -- CONT WITH  SCRATCH  MEDIA  ON
         xx0/
         where:  xx = DT, DK, MT, or MM
$CRSBF:: Internal crash stack
$CRSST== Top of crash stack
$CRSBN:: Starting device address
$CRSCS:: Checksum of device address
AGAIN:   Type a message and wait for the user
$CRSHT:: Wait for the user
$CRSUN:: Crash unit number
TYPE:    Type an ASCIZ message
CKSUM:   Verify checksum of device address
DUMP:    Dump the system image
```

Macro Library Calls -
HWDDF$ Define hardware registers

Entry Point -
$CRASH:: This is the system crash dump routine.

Calls TYPE, CKSUM

Input 02(SP) = PS word at crash.
(SP) = PC word at crash.

Output The internal crash stack and a core image of the system (up to 128K) are dumped onto the crash dump device.

7.2.4 CVRTM Module

CVRTM The CVRTM routine converts a pair of time interval-time units to a clock ticks count.

Macro Library Calls -
None

Entry Point -
$CVRTM:: Time interval-units pair to clock ticks count conversion

Calls $MUL

Input R3 = Address of the time interval pair.

Output CVRTM returns the ticks count to the calling routine by placing the high order part in R0 and the low order part in R1. CVRTM advances R3 by 4, thus pointing past the time interval-time units pair. If the calling routine specified an illegal time interval (greater than 15 bits) or illegal time units (0 or greater than 4), CVRTM returns a directive status of 'D.RS93'.

7.2.5 DRABO Module

DRABO The directive processing module, DRABO, terminates (aborts) the execution of a specified task.

Macro Library Calls -
ABODF$ Define task abort codes.

Entry Point -
$DRABO:: The DRABO routine aborts a specified task.

Calls $ABTSK

Input
R0 = Address of the TCB of the task to be terminated.
R1 = Address of the task status word of the task to be ended.
R2 = Address of the task status word of the current task.
R3 = Address of the last word in the DPB+2.
R4 = Address of the header of the current task.
R5 = Address of the TCB of the current task.

Output DRABO returns directive status (D.RS7) and PS to the task.

C = 0 if DRABO successfully completed execution. Also, DRABO returns a directive status of +1.

C = 1 if DRABO is rejected.

Directive status returned:
'D.RS7' if the specified task is not active

Note
DPB format:
WD. 00 -- DIC(83.),DPB size(3.)
WD. 01 -- First half of task name
WD. 02 -- Second half of task name

Error Reason for abort (in S.CABO) is set into R0 before calling $ABTSK.

7.2.6 DRASG Module

DRASG The directive processing module, DRASG, assigns a device unit to a logical unit number.

Macro Library Calls -
TCBDF$ Define task control block offsets.

Entry Point -
$DRASG:: Assigns logical unit number (LUN)

Calls $MPLUN, $MPLND, $MUL

Input R2 = Address of the task status word of the current task.
R3 = Address of the logical unit number in the DPB.
R4 = Address of the header of the current task.
R5 = Address of the TCB of the current task.

Output DRASG returns directive status and PS to the calling routine.

C = 0 if DRASG successfully completes execution. Also, DRASG returns a directive status of +1.

C = 1 if the DRASG directive routine is rejected.

Directive status returned:
'D.RS90' if a file is open or a unit is attached on the specified LUN.
'D.RS92' if a device or unit, or device and unit are invalid.

Note The DPB format is:
WD. 00 -- DIC(7.),DPB size(4.)
WD. 01 -- LUN to be assigned
WD. 02 -- Name of device to be assigned
WD. 03 -- Unit number of device to be assigned

7.2.7 DRATX Module

DRATX The directive processing module DRATX instructs the system to end the execution of an asynchronous system trap (AST) service routine. If another AST is queued and AST's are not disabled, the next queued AST is immediately executed.

Macro Library Calls -
ABODF$ Define task abort codes
HDRDF$ Define task header offsets
HWDDF$ Define hardware registers

Conditional Assembly Parameters -
A$$TRP AST support
C$$CKP Checkpointing support
D$$ISK Non-resident task support
M$$MGE Memory management
A$$CHK Address checking

Entry Point -
$DRATX:: The DRATX routine ends AST service.

Calls $NXTSK, $ACHCK, $RELOM, $SETRT, $ABCTK

Input
R2 = Address of the task status word of the current task
R3 = Address of the last word in the DPB +2
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRATX returns a directive status and PS word to the task.

C = 0 if the DRATX routine successfully completed execution. Also, the DRATX routine restores the status of the pre-AST state.

C = 1 if the DRATX routine is rejected.

Directive status returned:
'D.RS80' if an AST service routine did not execute the DRATX directive.

Note If an address check failure occurs while removing arguments from the task stack, the issuing task is aborted (S.CAST is placed in R0 and $ABCTK is called).

The DPB format is:
WD. 00 -- DIC(115.),DPB size(1.)

At issuance, the task stack contains:
14(SP) = (Unused)
12(SP) = (Unused)
10(SP) = (Unused)
06(SP) = Event flag mask word.
04(SP) = Pre-AST task PS
02(SP) = Pre-AST task PC
00(SP) = Pre-AST task directive status word

Error Possible ABORT as described in above note.

7.2.8 DRCIN Module

DRCIN The DRCIN module either connects or disconnects a specified interrupt vector to an interrupt service routine (ISR) in the task's own space.

Macro Library Calls -

HWDDF$	Define hardware registers
TCBDF$	Define task control block offsets
PCBDF$	Define partition control block offsets
ITBDF$	Define interrupt transfer block (ITB) offsets

Entry Point -
$DRCIN:: This routine connects a specified interrupt vector to an interrupt service routine (ISR) in the task's own space. DRCIN allocates a block of dynamic memory and sets up the block as an interrupt transfer block (ITB). The ITB is linked to the ITB list of the task with the listhead, a single word, in T.CPCB of the TCB. DRCIN disables shuffling and checkpointing for the task and sets up the vector to point to the offset X.JSR in the ITB that contains a subroutine call to the special interrupt save routine $INTSC.

Input

R2 = Address of the task status word of the current task
R3 = Pointer to WD. 01 in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output

DRCIN returns directive status and the PS to the task.

C = 0 if DRCIN successfully completes execution with a directive status of +1.

C = 1 if execution is unsuccessful.

Directive status returned:
D.RS1 An ITB could not be allocated (no pool space)
D.RS8 The function requested is disconnect and the task is not the owner of the vector.
D.RS16 Issuing task is not privileged
D.RS17 The specified vector is already in use
D.RS19 The specified vector is illegal (lower than 60 or higher than the highest vector specified at SYSGEN, or not a multiple of 4)
D.RS81 The ISR or disable-interrupt routine is not within 4K words from the specified base address

Note

The DPB format is:
WD. 00 -- DIC(129.), DPB size(7.)
WD. 01 -- Interrupt vector address
WD. 02 -- Base address for mapping of ISR and disable-interrupt routines. Ignored in an unmapped system.
WD. 03 -- Address of interrupt service routine. If zero, directive is "disconnect from interrupts" and remaining arguments are ignored.
WD. 04 -- Address of disable interrupt routine
WD. 05 -- (Low byte) low byte of PSW to be set before calling ISR
WD. 06 -- Address of AST routine

Entry Point -
$DISIN::

This routine disconnects a specified interrupt vector from an interrupt service routine (ISR) in the task's own space and deallocates the associated block of memory.

When disconnecting the last or only vector, DISIN clears the checkpoint-disable bit (T2.CKD) and the no-shuffle bit (PS.NSF). DISIN does this regardless of what the state was before vectors were connected or any change in state while vectors were connected except if the task is marked for abort. In this case it is not made shufflable.

The routine DISIN:

1. Removes the ITB from the ITB list starting in T.CPCB of the task's TCB
2. Calls the user routine that disables interrupts and that was supplied to DRCIN and to which the vector was connected.
3. Restores the vector PC to point to the nonsense interrupt routine

4. Removes the fork block of the ITB if it is in the fork queue
5. Removes the AST block of the ITB if it is in the AST queue for the task
6. Enables checkpointing for the task if this was the only vector connected to the task and makes the task shufflable if the task runs in a system controlled partition.
7. Deallocates the ITB

Input R1 = Pointer to the interrupt transfer block (ITB)
R5 = Pointer to task TCB

Output C = 0 if DISIN successfully completes execution

C = 1 if the task is not the owner of the vector

Note Registers R0, R1, R2, and R3 are altered by this routine

7.2.9 DRCMT Module

DRCMT DRCMT contains the following directive processing routines:

$DRCMT Cancel all mark time requests for the issuing task
$DRCSR Cancel all schedule requests for a specified task

Macro Library Calls -
CLKDF$ Define clock queue control block offsets

Entry Point -
$DRCMT:: The DRCMT routine cancels all mark time requests for the issuing task.

Calls None

Input R2 = Address of the task status word of the current task
R3 = Address of the last word in the DPB +2
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRCMT returns directive status and the PS to the task.

C = 0 with a directive status of +1

Note This routine is also called from the EXIT directive and requires only R5 to be loaded on entrance.
DRCMT shares common code with DRCSR.

The DPB format is:
WD. 00 -- DIC(27.),DPB size(1.)

Entry Point -
$DRCSR:: The DRCSR routine cancels all schedule requests for a specified task.

Calls $CLRMV - to remove periodic single-shot requests,

Input R0 = Address of the TCB for which to cancel schedule requests
R1 = Address of the task status word of the task for which to cancel schedule requests
R2 = Address of the task status word of the current task
R3 = Address of the last word in the DPB +2
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRCSR returns directive status and PS to the task.
C = 0 with a directive status of +1

Note The DPB format is:
WD. 00 -- DIC(25.),DPB size (3.)
WD. 01 -- First half of task name
WD. 02 -- Second half of task name

7.2.10 DRDAR Module

DRDAR The directive processing module, DRDAR, disables recognition of asynchronous system traps for the issuing task.

DRDAR contains the following directive processing routines:
$DRDAR Disable AST recognition
$DREAR Enable AST recognition

Macro Library Calls -
TCBDF$ Define task control block offsets

Entry Point -
$DRDAR:: The DRDAR routine disables AST recognition.

Calls None

Input R2 = Address of the task status word of the current task
R3 = Address of the last word in the DPB +2
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRDAR returns directive status and PS to the calling task.

C = 0 if the DRDAR directive successfully completes. Also, DRDAR returns a directive status of +1.

C = 1 if the DRDAR routine is rejected.

Directive status returned:
'D.RS8' if AST recognition is already disabled.

Note The DPB format is:
WD. 00 -- DIC(99.),DPB size(1.)

Entry Point -
$DREAR:: The DREAR routine causes the system to recognize asynchronous system traps for the issuing task. ASTs that have been queued while AST recognition was disabled are processed immediately.

Calls $SETRT

Input
R2 = Address of the task status word of the current task
R3 = Address of the last word in the DPB +2
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DREAR returns directive status and PS to the task.

C = 0 if the DREAR routine successfully completes. Also, DREAR returns a directive status of +1.

C = 1 if the DREAR routine is rejected.

Directive status returned:
'D.RS8' if AST recognition is not disabled.

Note
The DPB format is:
WD. 00 -- DIC(101.),DPB size (1.)

7.2.11 DRDCP Module

DRDCP The directive processing module, DRDCP, causes the system to disable or enable checkpointing of the issuing task.

DRDCP contains the following directive processing routines:
$DRDCP Disable checkpointing
$DRECP Enable checkpointing

Macro Library Calls -
TCBDF$ Define task control block offsets

Entry Point -
$DRDCP:: Disable checkpointing

Calls None

Input
R2 = Address of the task status word of the current task
R3 = Address of the last word in the DPB +2
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRDCP returns directive status and PS to the task.

C = 0 if the DRDCP routine successfully completes execution. Also, DRDCP returns a directive status of +1.

C = 1 if the DRDCP routine is rejected.

Directive status returned:
'D.RS8' if checkpointing is already disabled for the issuing task.
'D.RS10' if the issuing task is not checkpointable.

Note
The DPB format is:
WD. 00 -- DIC(95.),DPB size(1.)

Entry Point -
$DRECP:: The DRECP routine enables checkpointing of the issuing task.

Calls $NXTXK

Input
R2 = Address of the task status word of the current task
R3 = Address of the last word in the DPB +2
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRECP returns directive status and PS to the task.

C = 0 if the DRECP routine successfully completed execution. Also, DRECP returns a directive status of +1.

C = 1 if the DRECP routine is rejected.

Directive status returned:
'D.RS8' if checkpointing is already enabled.

Note
The DPB format is:
WD. 00 -- DIC(97.),DPB size(1.)

7.2.12 DRDSP Module

DRDSP The DRDSP module is the directive dispatcher.

DRDSP contains the following directive processing routines:

$TRTRP	The TRAP instruction traps to this routine
$EMTRP	The EMT instruction traps to this routine

Macro Library Calls -

HDRDF$	Define task header offsets
HWDDF$	Define hardware registers
TCBDF$	Define task control block offsets
WDBDF$	Define user window definition block offsets

Entry Point -
$TRTRP:: During execution, a TRAP instruction traps to this routine. TRTRP returns a directive status if the stack depth is zero. Otherwise, TRTRP transfers control to the EMT/TRAP synchronous system trap handling routine.

Calls None

Input
2(SP) = PS word pushed by TRAP instruction
(SP) = PC word pushed by TRAP instruction

Output TRTRP returns directive status in the stack if stack depth is zero.

Entry Point -

$EMTRP:: When an EMT instruction execution occurs, the trap occurs to this routine. EMTRP crashes the system if the stack depth is not +1. If the stack depth is +1, EMTRP checks the EMT instruction for the presence of code 377. If the code was other than 377, EMTRP transfers control to the EMT/TRAP SST handling routine. If 377 was present, EMTRP processes the coded directive.

The following processing occurs within EMTRP.

The EMTRP routine:

1. Calls $DIRSV (coroutine) to switch to system state and save registers R0 - R5.
2. Checks directive validity (DIC odd, size of DPB valid).
3. Maps to DPB using KAPR6 if a mapped system is being used.
4. Processes common functions if required (for instance, masks).
5. Sets up the following registers prior to calling the directive processing routine:
   R5 = Address of current task's TCB
   R4 = Address of current task's header
   R3 = Address of the next word in the directive DPB
   R2 = Address of second task status word of the current task
   R1 = Directive dependent
   R0 = Directive dependent
6. Calls the Directive Processing routine
7. Sets the DSW, gets correct stack pointers, and restores registers.
8. Enters $DIRXT through $DIRSV which restores R0 - R5 and exits.

Calls $ACHK2, $ACHKP, $CEFN, $SRSTD, $DIRSV, $SWSTK, $EMSST

Input 2(SP) = PS word pushed by the EMT instruction
(SP) = PC word pushed by the EMT instruction

Output EMTRP crashes the system if the stack depth is not +1. Otherwise, EMTRP gives control to the EMT/TRAP SST routine or the specified directive routine. If the EMT had a code of 377, EMTRP gives control to the specified directive routine.

Note The stack depth ($STKDP) is defined in SYSCM.

7.2.13 DREIF Module

DREIF This is the EXIT directive processing module.

DREIF contains the following directive processing routines:

$DREIF	End execution of issuing task if specified event flag is clear
$DREXT	End execution of issuing task

Macro Library Calls -

ABODF$	Define task abort codes
HDRDF$	Define task header offsets
PCBDF$	Define partition control block offsets
PKTDF$	Define I/O packet offsets
TCBCF$	Define task control block offsets

Entry Point -
$DREIF:: This routine causes the system to end the execution of the issuing task only if an indicated event flag is clear.

Calls None

Input
R0 = Event flag mask word
R1 = Event flag mask address
R2 = Address of the task status word of the current task
R3 = Address of the last word in the DPB+2
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DREIF returns directive status and the PS to the task.

C = 0 if DREIF successfully completes processing.

C = 1 if DREIF is rejected.

Directive status returned:
'D.RS22' if the specified event flag is set.
'D.RS97' if an invalid or no event flag number was specified when DREIF was called.

Note
The DPB format is:
WD. 00 -- DIC(53.),DPB size(2.)
WD. 01 -- Event flag number of event flag that must be clear

Entry Point -
DREXT:: The DREXT routine causes the system to end the execution of the issuing task.

Calls $DRCMT, $DRSIN, $RLMCB, $DASTT, $ABCTK, $IOKIL, $ALOCB, $DRQRQ, $ACTRM, $EXRQN, $TKWSE, $C5TA, $QMCRL, $RLPAR, $DETRG, $DEACB, $DRDSE, $QRMVF, $FINBF, $MPLNE,

Input
R2 = Address of the task status word of the current task
R3 = Address of the last word in the DPB +2
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

The dispatcher also calls this routine. When this occurs, only R5 need be loaded on entrance.

Output The DREXT routine returns directive status and the PS to the task.

No other status is returned to the task because this routine ends the task's execution.

Note
The DPB format is:
WD. 00 -- DIC(51.),DPB size(1.)

The DREXT routine contains a subroutine to empty a queue (MTQUE:) and a coroutine to scan a logical unit table (SCNLN:).

7.2.14 DREXP Module

DREXP The DREXP module extends the partition by a positive or negative increment.

Macro Library Calls -

HWDDF$	Define hardware offsets
HDRDF$	Define task header offsets
PCBDF$	Define partition control block offsets
TCBDF$	Define task control block offsets

Entry Point -
$DREXP:: The DREXP routine causes the system to extend the partition of the issuing task by a positive or negative amount.

Calls $SETRT, $CHKPT, $MAPTK

Input R2 = Address of the second task status word of the current task
R3 = Address of the extend increment in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output The DREXP routine returns directive status and the PS to the task.

C = 0 if the DREXP routine successfully completes processing. Also, DREXP returns a directive status of +1.

C = 1 if the DREXP routine is rejected.

Directive status returned:
'D.RS8' under the following conditions:

- The task is not checkpointable and specified a positive increment
- The task has a preallocated checkpoint space and is trying to extend its space greater than the installed size.
- The task is not in a system controlled partition.

'D.RS88' if the specified increment is invalid.

Note The DPB format is:
WD. 00 -- DIC(89.),DPB size(3.)
WD. 01 -- Extend increment
WD. 02 -- Reserved

7.2.15 DRGCL Module

DRGCL — The directive processing module, DRGCL, gets the MCR command line or releases the MCR command buffer.

DRGCL contains the following directive processing routines:

$DRGCL Transfer a 1 through 80 byte command line to MCR function task
$RLMCB Release MCR command buffer

Entry Point -
$DRGCL:: The DRGCL routine causes the system to transfer a 1 through 80. byte command line to the last MCR function task requested by the dispatcher.

Calls — Calls a subroutine to search for the command buffer for the current task.

Input — R2 = Address of the task status word of the current task
R3 = Address of the 80. byte buffer in the DPB.
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output — DRGCL returns the directive status and the PS to the current task.

C = 0 if DRGCL successfully completed execution. Also, DRGCL returns a directive status equal to the length of the command line in bytes.

C = 1 if DRGCL does not complete execution.

Directive status returned:
'D.RS80' if the issuing task is not the last task that was requested by the MCR dispatcher.

Note — The DPB format is:
WD. 00 -- DIC(127.),DPB size(41.)
WD. 01 through WD. 50 -- First through last word of the 80. byte buffer

Entry Point -
$RLMCB:: The RLMCB routine releases the MCR command buffer.

Calls — $DEACB, 30$ (Subroutine to search for the command buffer for the current task)

Input — R5 = Address of the TCB of the current task

Output — If the command line currently in the MCR command buffer is for the current task, RLMCB releases the buffer and clears $MCRTN.

7.2.16 DRGLI Module

DRGLI The directive processing module, DRGLI, causes the system to fill a six word buffer with information about a device that is assigned to a specified LUN. If requests to the device have been redirected, the information returned by DRGLI pertains to the redirected device.

Macro Library Calls -

HWDDF$	Define hardware registers

Entry Point -

$DRGLI:: Puts logical unit number information into a six word buffer.

Calls $MPLUN, $ACHKP, $DIV

Input

R2 = Address of the task status word of the current task
R3 = Address of the LUN in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output The DRGLI routine returns directive status and the PS to the task.

C = 0 if DRGLI successfully completes. Also, DRGLI returns a directive status of +1.

C = 1 if DRGLI does not complete exection.

Directive status returned:
'D.RS5' if no device is assigned to the specified LUN.

Note

The DPB format is:
WD. 00 -- DIC(5.),DPB size(3.)
WD. 01 -- LUN for which information is returned
WD. 02 -- Address of a six word buffer

The six-word buffer format is:
WD. 00 -- Name of assigned device
WD. 01 -- Unit number of assigned device and flags byte
WD. 02 -- First device characteristics word
WD. 03 -- Second device characterisitics word
WD. 04 -- Third device characteristics word
WD. 05 -- Fourth device characteristics word

7.2.17 DRGPP Module

DRGPP The directive processing module, DRGPP, causes the system to fill a 3-word buffer with partition parameters.

Macro Library Calls -

PCBDF$	Define partition control block offsets
TCBDF$	Define task control block offsets

Entry Point -

$DRGPP:: The directive processing routine, DRGPP, fills a 3-word buffer with partition parameters.

Calls $SRATT, $SRNAM, $ACHKP

Input R2 = Address of the task status word of the current task
R3 = Address of the partition name in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output The DRGPP routine returns directive status and the PS to the task.

C = 0 if DRGPP successfully completed execution. Also, DRGPP returns a directive status equal to the starting virtual address of the specified partition.

C = 1 if DRGPP is rejected.

Directive status returned:
'D.RS2' if the specified partition is not in the system.

Note The DPB format is:
WD. 00 -- DIC(65.),DPB size(4.)
WD. 01 -- First half of optional partition name
WD. 02 -- Second half of optional partition name
WD. 03 -- Address of a three word buffer

The buffer format is:
WD. 00 -- Base address of the partition in 32 word blocks
WD. 01 -- Size of the partition in 32 word blocks
WD. 02 -- Partition flags word

7.2.18 DRGSS Module

DRGSS The directive processing module, DRGSS, causes the system to store the contents of the console switch register in the issuing task's directive status word.

Macro Library Calls -
HWDDF$ Define hardware registers

Entry Point -
$DRGSS:: This routine gets the sense switch information.

Calls None

Input R2 = Address of the task status word of the current task
R3 = Address of the last word in the DPB +2
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output The DRGSS routine returns the directive status and the PS to the task.

C = 0 with a directive status equal to the contents of the console switch register.

Note The DPB format is:
WD. 00 -- DIC(125.),DPB size(1.)

7.2.19 DRGTK Module

DRGTK — The directive processing module, DRGTK, causes the system to fill a 16-word buffer with task parameters.

Macro Library Calls -

HDRDF$	Define task header offsets
PCBDF$	Define partition control block offsets
TCBDF$	Define task control block offsets

Entry Point -

$DRGTK:: This routine gets task parameters.

Calls None

Input

R2 = Address of the task status word of the current task
R3 = Address of the sixteen word buffer in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRGTK returns directive status and the PS to the task.

C = 0 with a directive status of +1.

Note

The DPB format is:
WD. 00 -- DIC(63.),DPB size(2.)
WD. 01 -- Address of a sixteen word buffer

The buffer format is:
WD. 00 -- First half of the name of the issuing task
WD. 01 -- Second half of the name of the issuing task
WD. 02 -- First half of the name of the task's partition
WD. 03 -- Second half of the name of the task's partition
WD. 04 -- First half of requesting task's name (not supported)
WD. 05 -- Second half of requesting task's name (not supported)
WD. 06 -- Task priority
WD. 07 -- Current task UIC
WD. 10 -- Number of logical units
WD. 11 -- Machine type indicator (not supported)
WD. 12 -- Standard flags word (not supported)
WD. 13 -- Address of task SST vector table
WD. 14 -- Size of task SST vector table in words
WD. 15 -- Size of task in bytes
WD. 16 -- Reserved
WD. 17 -- Reserved

7.2.20 DRGTP Module

DRGTP — The directive processing module, DRGTP, causes the system to fill a specified 8-word buffer with the current time parameters.

Entry Point -

$DRGTP:: The DRGTP routine fills an 8-word buffer with the current time parameters.

Macro Library Calls -
None

Calls $ACHKP

Input R2 = Address of the task status word of the current task
R3 = Address of the second word in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output The DRGTP routine returns directive status and the PS to the task.

C = 0 if the DRGTP routine successfully completes execution. Also, DRGTP returns a directive status of +1.

C = 1 if DRGTP does not complete execution.

Directive status returned:
'D.RS98' if the buffer is outside the issuing task's address space.

Note The DPB format is:
WD. 00 -- DIC(61.),DPB size(2.)
WD. 01 -- Address of an eight word buffer

The buffer format is:
WD. 00 -- Year since 1900
WD. 01 -- Month of year
WD. 02 -- Day of month
WD. 03 -- Hour of day
WD. 04 -- Minute of hour
WD. 05 -- Second of minute
WD. 06 -- Tick of second

7.2.21 DRMAP Module

DRMAP The directive processing module, DRMAP, contains the following routines:

$DRCRW::	Create an address window
$DRELW::	Eliminate address window
$DRMAP::	Map window to region
$DRUNM::	Unmap address window
$DRSRF::	Send by reference
$DRRRF::	Receive by reference

These directive processing routines receive, as input, pointers to a window definition block. The window definition block serves as a communication area between the issuing task and the Executive.

The format of the window definition block is:

W.NAPR W.NID	BASE APR	WINDOW ID
W.NBAS	VIRTUAL BASE ADDRESS (BYTES)	
W.NSIZ	WINDOW SIZE (32W BLOCKS)	
W.NRID	REGION ID	
W.NOFF	OFFSET IN PARTITION (32W BLOCKS)	
W.NLEN	LENGTH TO MAP (32W BLOCKS)	
W.NSTS	STATUS WORD	
W.NSRB	SEND/RECEIVE BUFFER ADDRESS (BYTES)	

Macro Library Calls -
HDRDF$ Define header and window block offset
PCBDF$ Define PCB and attachment descriptor offsets
TCBDF$ Define TCB offsets
WDBDF$ Define window definition block offsets

Entry Point -
$DRCRW:: The DRCRW routine causes the system to allocate an address window in the header of the issuing task. The routine unmaps and eliminates overlapping address windows and, optionally, maps the new window.

Calls ELAW (to eliminate address window)

Input

R2 = Address of the task status word of the current task
R3 = Address of the window definition block
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Input fields of the window definition block are:

W.NAPR = Base APR of region
W.NSIZ = Desired size of address window
W.NRID = ID of region to map or zero (0) for task region (if WS.MAP = 1)
W.NOFF = Offset within region to map (if WS.MAP = 1)
W.NLEN = Contains either length to map or zero (0). If zero, W.NLEN defaults to the smaller of the following: window size or size left in partition (if WS.MAP = 1)
W.NSTS = Control information
  WS.MAP = 1 if mapping is to occur
  WS.WRT = 1 if mapping is to occur with write access

Output

The DRCRW routine returns directive status and the PS to the task.

C = 0 if DRCRW successfully completes execution. Also, DRCRW returns a directive status of +1.

C = 1 if DRCRW does not complete execution.

Directive status returned:
'D.RS16' if the specified access is denied in the mapping stage of execution.
'D.RS84' if an invalid APR and window size combination or an invalid region and offset-length combination was specified in the mapping stage of execution.
'D.RS85' if no window blocks are available for use.
'D.RS86' if an invalid region was specified in the mapping stage of execution.

Output fields in the window definition block are:
W.NID = Assigned window ID
W.NBAS = Virtual base address of window
W.NLEN = Length actually mapped
W.NSTS = Indication of any changes in mapping status
  WS.CRW = 1 if address window is successfully established
  WS.ELW = 1 if any address windows were eliminated
  WS.UNM = 1 if any address windows were unmapped

Note

The DPB format is:
WD. 00 -- DIC(117.),DPB size(2.)
WD. 01 -- Address of window definition block

Entry Point -
$DRELW::

The DRELW routine causes the system to eliminate the specified address window, unmapping it first if necessary.

Calls

$SRWND, $UNMAP

Input

R2 = Address of the task status word of the current task
R3 = Address of the window definition block
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Input fields in the window definition block are:
W.NID = ID of address window to eliminate

Output

The DRELW routine returns directive status and the PS to the task.

C = 0 if DRELW successfully completed execution. Also, DRELW returns a directive status of +1 to the task.

C = 1 if DRELW is rejected.

Directive status returned:
'D.RS87' if an invalid address window was specified.

Output fields in the window definition block are:
W.NSTS = Indication of any changes in mapping status
  WS.ELW = 1 if DRELW successfully eliminated the address window
  WS.UNM = 1 if DRELW found the address window already unmapped

Note

The DPB format is:
WD. 00 -- DIC(119.),DPB size(2.)
WD. 01 -- Address of window definition block

Entry Point -
$DRMAP::

The DRMAP directive processing routine causes the system to map the specified address window to an offset in the specified region. DRMAP unmaps the window if necessary. DRMAP first builds an image of a mapped window block on the stack. If DRMAP encounters no errors during this process, DRMAP unmaps the corresponding window, if necessary, and sets up the new window from the stack image.

Calls

$SRWND, $SRATT, $UNMAP

Input

R2 = Address of the task status word of the current block
R3 = Address of the window definition block
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Input fields in the window definition block are:
W.NID = ID of window to be mapped
W.NRID = ID of region to which to map. If W.NRID is 0, mapping occurs to the task region.
W.NOFF = Offset within region to which to map
W.NLEN = Length to map. If 0, W.NLEN defaults to window size or the size left in the partition.
W.NSTS = Control information
  WS.WRT = 1 if write access is desired

Output

The DRMAP routine returns directive status and the PS to the task.

C = 0 if DRMAP successfully completes execution. Also, DRMAP returns a directive status of +1 to the task.

C = 1 if DRMAP is rejected.

Directive status returned:
'D.RS16' if the desired access to the region is denied.
'D.RS84' if an invalid region and offset size combination is specified.
'D.RS86' if an invalid region ID is specified.
'D.RS87' if an invalid address window is specified.

Output fields in the window definition block are:
W.NLEN = Length actually mapped
W.NSTS = Indication of any changes in mapping status.
  WS.UNM = 1 if the window was unmapped first

Note

The DPB format is:
WD. 00 -- DIC(121.),DPB size(2.)
WD. 01 -- Address of window definition block

Entry Point -
$DRUNM

The DRUNM routine causes the system to unmap the specified address window.

Calls

$SRWND, $UNMAP

Input

R2 = Address of the task status word of the current task
R3 = Address of the window definition block
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Input fields in the window definition block are:

W.NID = ID of the window to be unmapped

Output

DRNUM returns directive status and the PS to the task.

C = 0 if DRUNM successfully completes execution. Also, DRUNM returns a directive status of +1 to the task.
C = 1 if DRUNM is rejected.

Directive status returned:
'D.RS8' if the specified address window was not mapped.
'D.RS87' if an invalid address window was specified by the task.

Output fields in the window definition block are:
W.NSTS = Indicator of any changes in mapping status
  WS.UNM = 1 if DRUNM successfully unmapped the window

Note

The DPB format is:
WD. 00 -- DIC(123.),DPB size(2.)
WD. 01 -- Address of window definition block

Entry Point -
$DRSRF::

The DRSRF (Send by Reference) routine causes the system to create a formatted packet that includes a reference to a specified region and additional optional information which is supplied by the issuing task. The sender task must have the access specified in the reference. The referenced region is attached to the receiving task.

Calls

$CEFN, $ACHKP, $SRATT, $ALPKT, $CRATT, $QINSF, $DASTT, $DRDSE, $DEPKT

Input

R0 = Address of the TCB of the receiver task
R1 = Address of the task status word of the receiver task
R2 = Address of the task status word of the current task
R3 = Address of the EFN number in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Input fields in the window definition block are:
W.NRID = ID of the region to be sent by reference
W.NOFF = Offset word passed without checking
W.NLEN = Length word passed without checking
W.NSTS = Allowed access (defaults to access of sender task)
  WS.RED = 1 if read access is to be allowed
  WS.WRT = 1 if write access is to be allowed
  WS.EXT = 1 if extend access is to be allowed
  WS.DEL = 1 if delete access is to be allowed
W.NSRB = Optional address of an 8-word buffer of additional information

Output

The DRSRF routine returns directive status and the PS to the task.

C = 0 if DRSRF successfully completes execution. Also, DRSRF returns a directive status of +1.

C = 1 if the DRSRF routine did not complete execution.

Directive status returned:
'D.RS1' if DRSRF could not allocate a send packet or attachment descriptor
'D.RS2' if an attempt is made to send to an ACP task
'D.RS16' if the desired access to the region is denied
'D.RS86' if an invalid region ID was specified
'D.RS97' if an invalid EFN number is specified
'D.RS98' if the address check of the window definition block or send buffer fails

There are no output fields in the window definition block.

The format of the send by reference packet is:
WD. 00 -- Receive queue thread
WD. 01 -- TCB address if EFN was specified. Zero if the EFN was not specified.
WD. 02 -- EFN mask; first word of sender task name
WD. 03 -- EFN address; second word of sender task name
WD. 04 -- Region ID (attachement descriptor address)
WD. 05 -- Offset in region word
WD. 06 -- Length of map word
WD. 07 -- Status word
WD. 08 through WD. 017 -- Contents of the send buffer

Note

The DPB format is:
WD. 00 -- DIC(69.),DPB size(5.)
WD. 01 -- First half of receiver task name
WD. 02 -- Second half of receiver task name
WD. 03 -- Optional event flag to set when receive occurs
WD. 04 -- Address of the window definition block

Entry Point -
$DRRRF::

The DRRRF directive processing routine causes the system to dequeue the next receive by reference packet in the receive queue. DRRRF exits if there are no packets in the receive queue and the sending task requested an exit.

Calls

$DRDSE, $SETM, $ACHKP, $QRMVF, $DEPKT, $DRMAP

Input

R2 = Address of the task status word of the current task
R3 = Address of the window definition block
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Input fields in the window definition block are:
W.NSTS = Control information.
  WS.MSP = 1 if received reference is to be mapped
  WS.RCX = 1 if task exit desired if DRRRF does not find packet
W.NSRB = Optional address of 10-word buffer for additional information

Output

The DRRRF routine returns directive status and the PS to the task.

C = 0 if DRRRF successfully completes execution.

C = 1 if DRRRF is rejected.

Directive status returned:
'D.RS8' if receive by reference entry is not in the queue.
'D.RS98' if the address check of the receive buffer fails.

Output fields in the window definition block are:
W.NRID = Assigned region ID of the referenced region
W.NOFF = Offset word specified by sender task
W.NLEN = Length word specified by sender task
W.NSTS = Status word specified by sender task
  WS.RED = 1 if attached with read access
  WS.WRT = 1 if attached with write access
  WS.EXT = 1 if attached with extend access
  WS.DEL = 1 if attached with delete access
  WS.RRF = 1 if receive was successful

Note

The DPB format is:
WD. 00 -- DIC(81.),DPB size(2.)
WD. 01 -- Address of 10-word buffer for additional information

Entry Point -
$DRGMX::

The DRGMX directive processing routine causes the system to return the mapping context of the task (to fill in a given number of window definition blocks). The total number of window blocks is in the task header. DRGMX does not return information on unused window blocks.

Calls $ACHKP

Input R2 = Address of the task status word of the current task
R3 = Address of N window definition blocks
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

There are no input fields in the window definition blocks.

Output DRGMX returns directive status and the PS to the task.

C = 0 if DRGMX successfully completes execution. Also, DRGMX returns a directive status of +1.

C = 1 if DRGMX is rejected.

Directive status returned:
'D.RS98' if the address check of the window blocks plus terminator word fails.

Output fields in each window definition block are:
W.NID = Address window ID of next established address window.
W.NAPR = Base APR of the window
W.NBAS = Virtual base address of the window
W.NSIZ = Size of the address region
W.NRID = Region ID if mapped or unmodified
W.NOFF = Offset in region if mapped or unmodified
W.NLEN = Length of map if mapped or unmodified
W.NSTS = Necessary bits to restore mapping or 0 if not mapped
WS.MAP = 1 if window is mapped
WS.WRT = 1 if window is mapped with write access

Note The DPB format is:
WD. 00 -- DIC(113.),DPB size(2.)
WD. 01 -- Address of the N window definition blocks

7.2.22 DRMKT Module

DRMKT This module processes the MARK TIME and RUN directives.

DRMKT contains the following directive processing routines:

$DRMKT	Declare significant event at specified time interval
$DRRUN	Generate a clock queue entry to request a task at a specified time

Macro Library Calls -

CLKDF$	Define clock queue control block offsets

Entry Point -
$DRMKT:: The directive processing routine, DRMKT, causes the system to declare a significant event at a specified time interval from issuing the directive. If an event flag is specified at the time the MARK TIME directive is issued, DRMKT clears the event flag immediately and then sets the event flag at the time of the significant event.

If the issuing task specified an AST entry point, an asynchronous trap occurs at the time of the significant event. The PS, PC, directive status word, and the specified event flag number are pushed onto the stack when the specified AST is processed.

Calls $CVRTM

Input
R0 = Event flag mask word
R1 = Event flag mask address
R2 = Address of the task status word of the current task
R3 = Address of the third word in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRMKT returns directive status and the PS to the task.

C = 0 if DRMKT successfully completes execution. Also, DRMKT returns a directive status of +1.

C = 1 if DRMKT does not complete execution.

Directive status returned:
'D.RS1' if insufficient core is available to allocate the clock queue entry.

Note The DPB format is:

WD. 00 -- DIC(23.),DPB size(5.)
WD. 01 -- Event flag number of event flag to be set
WD. 02 -- Time interval magnitude
WD. 03 -- Time interval units
WD. 04 -- AST entry point address

Entry Point -
$DRRUN:: The DRRUN directive processing routine causes the system to generate a clock queue entry to cause a task to be requested at a specified delta time from issuance of the directive and, optionally, to repeat the request periodically.

Calls $UISET, $CVRTM, $ALCLK, $CLINS

Input
R0 = Address of the TCB of the task to be run
R1 = Address of the task status word of the task to be run
R2 = Address of the task status word of the current task
R3 = Address of the partition name in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output — The routine DRRUN returns directive status and the PS to the task.

C = 0 if DRRUN successfully completes execution. Also, DRRUN returns a directive status of +1 to the task.

C = 1 if DRRUN does not complete execution.

Directive status returned:
'D.RS1' if insufficient core is available to allocate the clock queue entry.

Note — The DPB format is:

WD. 00 -- DIC(17.),DPB size (11.)
WD. 01 -- First half of task name
WD. 02 -- Second half of task name
WD. 03 -- Partition name (not supported but must be present)
WD. 04 -- Partition name (not supported but must be present)
WD. 05 -- Request priority (not supported but must be present)
WD. 06 -- Request UIC
WD. 07 -- Delta time magnitude
WD. 10 -- Delta time units
WD. 11 -- Reschedule interval magnitude
WD. 12 -- Reschedule interval units

7.2.23 DRPUT Module

DRPUT — DRPUT contains the following directive processing routines:

$DRFEX	Enable or disable floating-point ASTs for task
$DRPUT	Enable or disable power recovery ASTs for task
$DRRRA	Enable or disable receive-by-reference ASTs for task
$DRRCV	Enable or disable receive ASTs for task

Macro Library Calls -

HDRDF$	Define task header offsets
TCBDF$	Define task control block offsets

Entry Point -
$DRFEX:: — The directive processing routine, DRFEX, causes the system to record that floating-point ASTs are either desired or not desired for the issuing task.

Calls — None

Input — R2 = Address of the task status word of the current task
R3 = Address of the AST address in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRFEX returns directive status and the PS to the task.

C = 0 if DRFEX successfully completed execution. Also, DRFEX returns a directive status of +1 to the task.

C = 1 if DRFEX does not complete execution.

Directive status returned:
'D.RS8' if ASTs are already not desired
'D.RS80' if an AST routine issued this directive

Note The DPB format is:
WD. 00 -- DIC(111.),DPB size(2.)
WD. 01 -- AST entry point address or zero

DRFEX shares common code with DRRCV.

Entry Point -
$DRPUT:: The directive processing routine, DRPUT, causes the system to record that power recovery ASTs either are desired or are not desired for the issuing task.

Input R2 = Address of the task status word of the current task
R3 = Address of the AST address in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRPUT returns directive status and the PS to the task.

C = 0 if DRPUT succesfully completes execution. Also, DRPUT returns a directive status of +1 to the task.

C = 1 if DRPUT does not complete execution.

Directive status returned:
'D.RS8' if ASTs are already not desired
'D.RS80' if an AST routine issued the DRPUT directive.

Note The DPB format is:
WD. 00 -- DIC(109.),DPB size(2.)
WD. 01 -- AST entry point address or zero

DRPUT shares common code with DRRCV.

Entry Point -
$DRRRA:: The directive processing routine, DRRRA, causes the system to record that receive-by-reference ASTs either are desired or are not desired for the issuing task.

Calls None

Input R2 = Address of the task status word of the current task
R3 = Address of the AST address in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRRRA returns directive status and the PS to the task.

C = 0 if DRRRA successfully completes execution. Also, DRRRA returns a directive status of +1 to the task.

C = 1 if DRRRA does not complete execution.

Directive status returned:
'D.RS8' if ASTs are already not desired.
'D.RS80' if an AST routine issued the DRRRA directive

Note The DPB format is:
WD. 00 -- DIC(21.),DPB size(2.)
WD. 01 -- AST entry point address or zero

Entry Point -
$DRRCV:: The directive processing routine, DRRCV, causes the system to record that receive ASTs either are desired or are not desired for the issuing task.

Calls $ALCLK, $DECLK

Input R2 = Address of the task status word of the current task
R3 = Address of the AST address in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRRCV returns directive status and the PS to the task.

C = 0 if DRRCV successfully completed. Also, DRRCV returns a directive status of +1 to the task.

C = 1 if DRRCV does not complete execution.

Note The DPB format is:
WD. 00 -- DIC(107.),DPB size(2.)
WD. 01 -- AST entry point address or zero

7.2.24 DRQIO Module

DRQIO DRQIO contains the following directive processing routines:

$DRQIO	Place I/O request in a queue of priority ordered requests
$DRGRG	Place I/O packet in a controller queue

Macro Library Calls -

F11DF$	Define Files-11 control block offsets
HWDDF$	Define hardware registers
PKTDF$	Define I/O packet offsets
TCBDF$	Define task control block offsets
PCBDF$	Define partition control block offsets

Entry Point -
$DRQIO:: The directive processing routine, DRQIO, places an I/O request in a queue of priority ordered requests for a device or unit specified by a logical unit number. If the task specifies an event flag with a QIO and WAIT (QIOW) directive, the task is put into a wait state to wait for the specified event flag to be set upon the occurrence of the significant event.

Calls $MPLUN, $TKWSE, $CEFN, $ACHKW, $ALPKT, $CEFI, $DRWFS, $RELOC, $ACHKB, $MPPHY, $IOKIL, $ACHCK, $MPLND

Input
R2 = Address of the task status word of the current task
R3 = Address of the I/O function code in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRQIO returns directive status and the PS to the task.

C = 0 if DRQIO successfully completes execution. Also, DRQIO returns a directive status of +1 to the task.

C = 1 if DRQIO is rejected.

Directive status returned:
'D.RS5' if the specified LUN is not assigned.

Note
The DPB format is:
WD. 00 -- DIC(1./3.),DPB size(12.)
WD. 01 -- I/O function code
WD. 02 -- LUN and unused byte
WD. 03 -- Event flag number and priority (priority is ignored)
WD. 04 -- Address of I/O status block
WD. 05 -- Address of AST service routine
WD. 06 -- Parameter 1
WD. 07 -- Parameter 2
WD. 10 -- Parameter 3
WD. 11 -- Parameter 4
WD. 12 -- Parameter 5
WD. 13 -- Parameter 6

Entry Point -
$DRQRQ:: The DRQRQ routine is called to insert an I/O packet in a controller queue and call the driver to start activity on the device.

Calls $QINSP, @D.VINI(R5) call to driver initiator or device initiator, $DEPKT, $IOFIN, PPRM, $ALOCB, $ACHCK, $RELOM, RQPRM

Input
R1 = Address of the I/O packet
R5 = Address of the unit control block

Output DRQRQ places the I/O packet in the controller queue and starts activity on the device.

Note This routine destroys the contents of R4.

7.2.25 DRRAS Module

DRRAS DRRAS contains the following directive processing routines:
$DRREC Process RECEIVE DATA and RECEIVE DATA OR EXIT directives.
$DRSND Process SEND DATA directive.

Macro Library Calls -
HDRDF$ Define task header offsets
TCBDF$ Define task control block offsets

Entry Point -
$DRREC:: This routine causes the system to dequeue a data block from the issuing task's receive queue. If the issued directive was RECEIVE DATA OR EXIT, the task exits if no data is queued.

Calls $ACHKP, $QRMVF, $DEPKT

Input
R2 = Address of the task status word of the current task
R3 = Address of the second word in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output The DRREC routine returns directive status and the PS to the task.

C = 0 if DRREC successfully completes execution. Also, DRREC returns a directive status of +1 to the task.

C = 1 if DRREC does not complete execution.

Directive status returned:
'D.RS8' if no data is queued in the task's receive queue.

Note The DPB format is:
WD. 00 -- DIC(75. or 77.),DPB size(4.)
WD. 01 -- First half of the task name - Not supported but must be present
WD. 02 -- Second half of the task name - Not supported but must be present
WD. 03 -- Address of a fifteen word receive buffer

Entry Point -
$DRSND:: This directive processing routine causes the system to queue a thirteen word data block in a specified task's receive queue.

Calls $ACHKP, $SETF, $ALPKT, $QINSF, $DASTT, $DRDSE

Input
R0 = Address of the TCB of the receiver task
R1 = Address of the task status word of the receiver task
R3 = Address of the data block address in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRSND returns directive status and the PS to the task.

C = 0 if DRSND successfully completes execution. Also, DRSND returns a directive status of +1 to the task.

C = 1 if DRSND does not complete execution.

Directive status returned:
'D.RS1' if insufficient core is available to queue the data block.
'D.RS2' if the receiver task is an ancillary control processor.

Note The DPB format is:
WD. 00 -- DIC(71.),DPB size(5.)
WD. 01 -- First half of receiver task name
WD. 02 -- Second half of receiver task name
WD. 03 -- Address of thirteen word data block
WD. 04 -- Event flag number (optional)

7.2.26 DRREG Module

DRREG The directive processing module, DRREG, contains the following routines:

$DRCRR::	Create a region
$DRATR::	Attach a region
$DRDTR::	Detach a region
$DETRG::	Detach a region by attachment descriptor address

These directive processing routines receive, as input, a pointer to a region definition block that is a communication area between the issuing task and the Executive.

The format of the region definition block is:

R.GID	REGION ID
R.GSIZ	SIZE OF REGION (32W BLOCKS)
R.GNAM	NAME OF REGION (RAD50)
R.GPAR	REGION'S MAIN PARTITION NAME (RAD50)
R.GSTS	REGION STATUS WORD
R.GPRO	PROTECTION CODE OF REGION

Macro Library Calls -

HDRDF$	Define header and window block offsets
PCBDF$	Define PCB and attachment descriptor offsets
RDBDF$	Define region definition block offsets
TCBDF$	Define TCB offsets

Entry Point -

$DRCRR:: This directive processing routine causes the system to create a region and optionally to attach it.

Calls $SRNAM, ATT, $ALOCB, $FNDSP

Input

R2 = Address of the task status word of the current task
R3 = Address of the region definition block
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Input fields in the region definition block are:
R.GSIZ = Size of region to create
R.GNAM = Name of region to create or 0 for no name
R.GPAR = Name of system partition in which to allocate region or zero (0) for main system partition of task
R.GSTS = Control information
  RS.NDL = 1 if region should not be deleted on last detach
  RS.ATT = 1 if created region should be attached
  RS.RED = 1 if read access is desired on attach
  RS.WRT = 1 if write access is desired on attach
  RS.EXT = 1 if extend access is desired on attach
  RS.DEL = 1 if delete access is desired on attach
R.GPRO = Protection code for region (DEWR,DEWR,DEWR,DEWR)

Output

DRCRR returns directive status and the PS to the task.

C = 0 if DRCRR successfully completed execution. Also, DRCRR returns a directive status of +1 to the task.

C = 1 if DRCRR does not complete execution.

Directive status returned:
'D.RS1' if a PCB or attachement descriptor could not be allocated
'D.RS16' if the desired access is denied in the attachement stage
'D.RS84' if the specified partition in which the region is to be allocated does not exist, or if no partition name has been specified and RS.ATT is zero.

Output fields in the region definition block are:
R.GID = Assigned region ID (RS.ATT = 1)
R.GSTS = Directive completion information
  RS.CRR = 1 if region was created

Note

The DPB format is:
WD. 00 -- DIC(55.),DPB size(2.)
WD. 01 -- Address of region definition block

Entry Point -
$DRATR:: The directive processing routine, DRATR, causes the system to attach the specified region to the current task.

Calls $SRNAM, $CKACC, $CRATT

Input R2 = Address of the task status word of the current task
R3 = Address of the region definition block
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Input fields in the region definition block are:
R.GNAM = Name of the region to which to attach or zero (0) for task region
R.GSTS = Desired access to region
  RS.RED = 1 if read access is desired
  RS.WRT = 1 if write access is desired
  RS.EXT = 1 if extend access is desired
  RS.DEL = 1 if delete access is desired

Output DRATR returns directive status and the PS to the task.

C = 0 if DRATR successfully completes execution. Also, DRATR returns a directive status of +1 to the task.

C = 1 if DRATR does not complete execution.

Directive status returned:
'D.RS1' if an attachment descriptor cannot be allocated.
'D.RS84' if the specified region name does not exist.

Output fields in the region definition block are:
R.GID = Assigned region ID
R.GSIZ = Size of attached region

Note The DPB format is:
WD. 00 -- DIC(57.),DPB size(2.)
WD. 01 -- Address of region definition block

Entry Point -
DRDTR:: The directive processing routine, DRDTR, causes the system to detach the specified region, unmapping it if necessary.

Calls $SRATT, $UNMAP

Input R2 = Address of the task status word of the current task
R3 = Address of the region definition block
R4 = Address of the headee of the current task
R5 = Address of the TCB of the current task

Input fields in the region definition block are:
R.GID = Region ID of the region to be detached
R.GSTS = Control information
  RS.MDL = 1 if region should be marked for delete on the last detach

Output — DRDTR returns directive status and the PS to the task.

C = 0 if DRDTR successfully completes execution. Also, DRDTR returns a directive status of +1 to the task.

C = 1 if DRDTR does not complete execution.

Directive status returned:
'D.RS16'if an attempt is made to mark the region for delete without delete access
'D.RS86' if an invalid region ID is specified or if an attempt is made to detach region zero (0).

Output fields in the region definition block are:
R.GSTS = Indication of any changes in mapping context
  RS.UNM = 1 if any windows were unmapped

Entry Point -
$DETRG:: — The directive processing routine, DETRG, detaches a task from a region and deallocates the attachment descriptor. The last time DETRG detaches the region it checks it for deletion and calls $NXTSK if needed.

Calls — $QRMVT, $RLPR1, $DEACB

Input — R5 = Address of attachment descriptor

Output — DETRG modifies R0, R1, R2, and R3.

All other output is the same as the DRDTR routine.

7.2.27 DRREQ Module

DRREQ — The directive processing module, DRREQ, causes the system to request the execution of a specified task.

Macro Library Calls -
TCBDF$ — Define task control block offsets

Entry Point -
$DRREQ:: — Request the execution of a specified task.

Calls — $TSKRP, $UISET

Input
R0 = Address of the TCB of the task to be requested
R1 = Address of the task status word of the task to be requested
R2 = Address of the task status word of the current task
R3 = Address of the partition name in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRREQ returns directive status and the PS to the task.

C = 0 if DRREQ successfully completes execution. Also, DRREQ returns directive status of +1 to the task.

C = 1 if DRREQ does not complete execution.

Directive status returned:
'D.RS1' if partition control block cannot be allocated
'D.RS7' if specified task is already active

Note The DPB format is:
WD. 00 -- DIC(11.),DPB size(7.)
WD. 01 -- First half of task name
WD. 02 -- Second half of task name
WD. 03 -- Partition name (not supported, but must be present)
WD. 04 -- Partition name (not supported, but must be present)
WD. 05 -- Request priority (not supported, but must be present)
WD. 06 -- Request UIC

7.2.28 DRRES Module

DRRES The directive processing module, DRRES, contains the following directive processing routines:

$DRRES	Resume execution of a task that has issued a suspend directive
$DRSPN	Suspend the execution of the task that issued this directive
$DRATP	Change the task priority of the specified task

Macro Library Calls -

HDRDF$	Define task header offsets
PKTDF$	Define I/O packet offsets
PCBDF$	Define partition control block offsets
TCBDF$	Define task control block offsets

Entry Point -
$DRRES:: The directive processing routine, DRRES, causes the system to resume the execution of a task that issued the suspend directive.

Calls $SETCR

Input R0 = Address of the TCB of the task to be resumed
R1 = Address of the task status word of the task to be resumed
R2 = Address of the task status word of the current task
R3 = Address of the last word in the DPB+2
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output

DRRES returns directive status and the PS to the task.

C = 0 if DRRES successfully completes execution. Also, DRRES returns a directive status of +1 to the task.

C = 1 if DRRES is rejected.

Directive status returned:
'D.RS7' if the specified task is not active
'D.RS8' if the specified task is not suspended

Note

The DPB format is:
WD. 00 -- DIC(47.),DPB size(3.)
WD. 01 -- First half of task name
WD. 02 -- Second half of task name

Entry Point -
$DRSPN::

The directive processing routine, DRSPN, causes the system to suspend the execution of the issuing task.

Calls

$SETRT

Input

R2 = Address of the task status word of the current task
R3 = Address of the last word in the DPB+2
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output

DRSPN returns directive status and the PS to the task.

C = 0 with a directive status of 'D.RS22'

Note

WD. 00 -- DIC(45.),DPB size(1.)

Entry Point -
$DRATP::

The directive processing routine, DRATP, causes the system to change the task priority of the specified task.

Calls

$MPLNE, $ACTRM, $ACTTK, $QRMVT, $NXTSK, $DRDSE, $QINSF, $QRMVF, $QINSP

Input

R0 = Address of the TCB of the task to be altered
R1 = Address of the task status word of the task to be altered
R2 = Address of the task status word of the current task
R3 = Address of the last word in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output

DRSPN returns directive status and the PS to the task.

C = 0 if DRSPN successfully completes execution. Also, DRSPN returns directive status of +1 to the task.

C = 1 if DRSPN does not complete execution.

Directive status returned:
'D.RS7' if the task is not active
'D.RS95' if the new specified priority is invalid

Note

The DPB format is:
WD. 00 -- DIC(9.),DPB size(4.)
WD. 01 -- First half of task name
WD. 02 -- Second half of task name
WD. 03 -- New priority

7.2.29 DRSED Module

DRSED The directive processing module, DRSED, contains the following directive processing routines:

$DRCEF	Clear event flag
$DRDSE	Declare a significant event
$DRRAF	Read all event flags
$DRSEF	Set event flag
$TKWSE	Task wait for significant event
$DRWSE	Wait for significant event
$DRWFL	Wait for LOGICAL OR of event flags
$DRWFS	Wait for single event flag

Macro Library Calls -

HDRDF$	Define task header offsets
TCBDF$	Define task control block offsets

Entry Point -

$DRCEF:: The directive processing routine, DRCEF, causes the system to report the polarity of an event flag and then clear the event flag.

Calls None

Input
R0 = Event flag mask word
R1 = Event flag mask address
R2 = Address of the task status word of the current task
R3 = Address of the last word in the DPB+2
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRCEF returns directive status and the PS to the task.

C = 0 with a directive status of 'D.RS00' if the flag was clear or 'D.RS22' if the flag was set.

Note The DPB format is:
WD. 00 -- DIC(31.),DPB size(2.)
WD. 01 -- Event flag number of flag to be cleared

Entry Point -

$DRDSE:: The directive processing routine, DRDSE, causes the system to declare a significant event.

This directive is also called as a subroutine.

Calls None

Input
R2 = Address of the task status word of the current task
R3 = Address of the last word in the DPB+2
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRDSE returns directive status and the PS to the task.

C = 0 with a directive status of +1

Note The DPB format is:
WD. 00 -- DIC(35.),DPB size(1.)

Entry Point -
$DRRAF:: The directive processing routine, DRRAF, causes the system to fill a 4-word buffer with the task's local and the global event flags.

Calls $ACHKP

Input
R2 = Address of the task status word of the current task
R3 = Address of the buffer address in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRRAF returns directive status and the PS to the task.

C = 0 if DRRAF successfully completes execution. Also, DRRAF returns a directive status of +1 to the task.
C = 1 if DRRAF does not complete execution.

Directive status returned:
'D.RS98' if the buffer is outside of the issuing task's address space

Note
The DPB format is:
WD. 00 -- DIC(39.),DPB size(2.)
WD. 01 -- Address of a 4-word buffer

Entry Point -
$DRSEF:: The directive processing routine, DRSEF, causes the system to report on the polarity of an event flag and then set the event flag.

Calls None

Input
R0 = Event flag mask word
Rl = Event flag mask address
R2 = Address of the task status word of the current task
R3 = Address of the last word in the DPB+2
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRSEF returns directive status and the PS to the task.

C = 0 with a directive status of 'D.RS00' if the flag was clear or 'D.RS22' if the flag was set

Note
The DPB format is:
WD. 00 -- DIC(33.),DPB size(2.)
WD. 01 -- Event flag number of flag to be set

Entry Point -
$TKWSE:: This routine is called from within the Executive to execute a wait for significant event directive for the current task. This routine shares code that is common with the $DRWSE routine.

Calls None

Input None

Output This routine executes the wait for significant event directive and returns to the calling routine.

Entry Point -
$DRWSE:: The directive processing routine, DRWSE, causes the system to suspend the execution of the issuing task until the next significant event occurs.

Calls $SETRQ

Input
R2 = Address of the task status word of the current task
R3 = Address of the last word in the DPB+2
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRWSE returns directive status and the PS to the task.

C = 0 with a directive status of +1

Note
The DPB format is:
WD. 00 -- DIC(49.),DPB size(1.)

Entry Point -
$DRWFL:: The directive processing routine, DRWFL, causes the system to suspend the execution of the task that issued the directive until any of the specified event flags become set.

Calls None

Input
R2 = Address of the task status word of the current task
R3 = Address of the second word in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output DRWFL returns directive status and the PS to the task.

C = 0 if DRWFL successfully completes execution. Also, DRWFL returns a directive status of +1 to the task.

C = 1 if DRWFL does not complete execution.

Directive status returned:
'D.RS97' if an illegal event flag set or a zero (0) event flag mask is specified by the task.

Note
The DPB format is:
WD. 00 -- DIC(43.),DPB size(3.)
WD. 01 -- Event flag set indicator
WD. 02 -- Event flag mask word

The event flag sets are:
Set 0 -- Event flags 1. - 16.
Set 1 -- Event flags 17. - 32.
Set 2 -- Event flags 33. - 48.
Set 3 -- Event flags 49. - 64.

Entry Point -
$DRWFS:: The directive processing routine, DFWFS, causes the system to suspend the execution of the issuing task until a specified event flag is set.

Calls $SETRT

Input

R0 = Event flag mask word
R1 = Event flag mask address
R2 = Address of the task status word of the current task
R3 = Address of the last word in the DPB+2
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output

The DRWFS routine returns directive status and the PS to the task.

C = 0 with a directive status of +1.

Note

The DPB format is:
WD. 00 -- DIC(41.),DPB size(2.)
WD. 01 -- Event flag number of flag to wait for

7.2.30 DRSST Module

DRSST

The directive processing module, DRSST, specifies SST vectors of service routine entry points for use by intra-task debugging aids or the issuing task.

DRSST contains the following directive processing routines:

$DRSDV	Record address and length of a vector of SST service routine entry points for debugging aid.
$DRSTV	Record address and length of a vector of SST service routine entry points for issuing task.

Macro Library Calls -

HDRDF$	Define task header offsets

Entry Point -

$DRSDV::

The directive processing routine, DRSDV, records the address and length of a vector of SST service routine entry points for use by an intra-task debugging aid (ODT).

Calls

None

Input

R2 = Address of the task status word of the current task
R3 = Address of the second word in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output

DRSDV returns directive status and the PS to the task.

C = 0 if DRSDV successfully completes execution. Also, DRSDV returns a directive status of +1 to the task.

C = 1 if DRSDV does not complete execution.

Directive status returned:

'D.RS98' if part of the vector is outside of the issuing task's address space, a vector address of zero is specified, or the vector size is greater than 31. words.

Note

The DPB format is:
WD. 00 -- DIC(103.),DPB size(3.)
WD. 01 -- Address of the SST vector
WD. 02 -- Number of entries in the SST vector

The SST vector format is:
WD. 00 -- Traps to 4 (odd address, non-existent memory, etc.)
WD. 01 -- Segment fault
WD. 02 -- Trace trap (T-bit) or execution of BPT instruction
WD. 03 -- Execution of an IOT instruction
WD. 04 -- Execution of an illegal or reserved instruction
WD. 05 -- Execution of a non-RSX EMT instruction
WD. 06 -- Execution of a TRAP instruction
WD. 07 -- PDP 11/40 floating point exception fault

Entry Point -
$DRSTV::

The directive processing routine, DRSTV, causes the system to record tha address and length of a vector of SST service routine entry points for use by the issuing task.

Calls

$ACHKW

Input

R2 = Address of the task status word of the current task
R3 = Address of the second word in the DPB
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output

DRSTV returns directive status and the PS to the task.

C = 0 if DRSTV successfully completes execution. Also, DRSTV returns a directive status of +1 to the task.

C = 1 if DRSTV does not complete execution.

Directive status returned:
'D.RS98' if part of the vector is outside of the issuing task's address space, a vector address of zero is specified, or the vector size is greater than 31. words.

Note

The DPB format is:
WD. 00 -- DIC(105.),DPB size(3.)
WD. 01 -- Address of the SST vector
WD. 02 -- Number of entries in the SST vector

The SST vector format is:
WD. 00 -- Traps to 4 (odd address, non-existent memory, etc.)
WD. 01 -- Segment fault
WD. 02 -- Trace trap (T-bit) or exectution of a BPT instruction
WD. 03 -- Execution of an IOT instruction
WD. 04 -- Execution of an illegal or reserved instruction
WD. 05 -- Execution of a non-RSX EMT instruction
WD. 06 -- Execution of a TRAP instruction
WD. 07 -- PDP 11/40 floating-point exception fault

7.2.31 ERROR Module

ERROR This is the error logging module. This module contains the following routines:

$ALEMB - Allocate an error message block

$ALEB1 - Allocate an error message block (alternate entry)

$BMSET - Set a driver's bit in the I/O active bit map

$DTOER - Device timeouts

$DVCER - Device error bit set

$DVERR - Device error bit set (temporary label)

NSIER: - Nonsense interrupt error processing

$QEMB - Queue an error message block (EMB)

Macro Library Calls -
HWDDF$ Define CPU registers
CLKDF$ Define clock offsets and codes
HDRDF$ Define task header offsets
PCBDF$ Define partition offsets
PKTDF$ Define I/O packet offsets
TCBDF$ Define task control block offsets and codes

Entry Point -
$ALEMB::
$ALEB1:: Error servicing routines call this routine. It counts the occurence of the error and tries to allocate a core block from the pool. If the core block is allocated, it fills in the error code, the time, and the error sequence number. Otherwise, it sets the C-bit = 1.

Calls $ALOCB

Input 2(SP) = Error code
0(SP) = Return
R1 = Size of the EMB to allocate

Output If the C-bit is 0:
(R0) = Address of the first unfilled byte
(R1) = Address of the EMB

If the C-bit = 1, $ALEMB did not complete execution.

Note $ALEMB destroys R2 and R3 when it calls $ALOCB.

Entry Point -
$BMSET:: This co-routine raises the processor priority to seven and sets the mask in the SCB in $IOABM. It lets the calling routine start the function, then allows interruptions.

Calls @(SP)+ -- to call the calling routine

Input R4 = Address of the SCB

Output $IOABM is modified and priority 7 established.

Entry Point -
$DTOER:: This is the error message block (EMB) formatting routine. The driver recognizes timeout errors. On the first occurrence of an error, $DTOER attempts to log it. If errors occur on retries, they are not logged.

$DTOER pushes the error code EC.DTO on the stack, sets the error in progress bit in the SCB, calculates the length of the required EMB, and calls $ALEMB. If $ALEMB fails to allocate a packet for any reason, $DTOER exits and $DVCER clears the pointer in the SCB to the EMB.

Otherwise, $DVCER copies the saved $IOABM from the SCB to the EMB and saves a pointer to the EMB in the SCB. $DVCER puts the error information, including device registers, into the EMB and executes a RETURN. The contents of the CSR that is saved is unchanged from the time of timeout. After the CSR is saved, device interrupts are disabled and CPU priority is lowered to PR0.

Calls None

Input (R2) = Address of the CSR
(R4) = Address of the SCB

Output C = 0 if the function was not a user-mode diagnostic function. The EMB is filled and the SCB contains a pointer to it. The error in progress flag is set in the SCB.

C = 1 if the function was a user-mode diagnostic function. Only interrupt enable is cleared and the priority is lowered to 0.

Note If the system supports diagnostics, R1 will be set to the I/O packet address. If diagnostics are not supported, all registers are cleared.

Entry Point -
$DVCER:: This is the EMB formatting routine and it is used when the device driver recognizes device error bit errors. On the first occurence of an error, $DVCER attempts to log it. If errors occur on retries, they are not logged.

$DVCER pushes the error code, EC.DVC, on the stack, sets the error in progress bit in the SCB, calculates the length of the required EMB, and calls $ALEMB. If $ALEMB fails to allocate a packet for any reason, $DVCER clears the pointer in the SCB to the EMB and exits.

Otherwise, $DVCER copies the saved $IOABM from the SCB to the EMB and saves a pointer to the EMB in the SCB. $DVCER puts the error information, including the device registers, into the EMB and executes a RETURN.

Calls $ALEMB

Input (R4) = Address of the SCB

After $DVCER fills the stack, the stack contains:

0(SP) = Error code
2(SP) = CSR address or 0
4(SP) = Saved R0
6(SP) = Saved R1
10(SP) = Saved R2
12(SP) = Saved R3
14(SP) = Return

Output If successful, the EMB is filled and the SCB contains a pointer to it. An error in progress bit is set in the SCB.

Otherwise, the occurrence of the error is counted only.

Entry Point -
$NS0::through$NS7::
These are the nonsense interrupt identifier routines. One of a group of 16 unused vectors points to each of these routines. The vectors are sub-coded in the PS condition codes. Each routine consists of a CALL to NSIER: and a word containing an indentifying number.

Entry Point -
$QEMB:: This is the common entry point for all EMBs. $QEMB queues the EMB FIFO in the error queue. $QEMB awakens the ERROR logger task if there are enough bytes of EMBs in the pool. If the queue is empty, $QEMB makes a schedule request to write a queued EMB within a time limit. Otherwise, $QEMB executes a RETURN.

Input (R1) = Address of the EMB

Output None

Note $QEMB destroys registers R0 through R3

7.2.32 INITL Module

INITL The INITL module contains the transfer point of the resident executive. When the system is initially booted, control transfers to this routine to initialize and start up the system.

The INITL module contains the following labels:

$POOL::	Start of the system pool space
SYSMG:	The log on message "RSX-11M V3.1 BL",
SYSID:	The system identification (4 bytes)
DEVMG:	The message: /DEVICE dduu: NOT IN CONFIGURATION/
OPMSG:	Subroutine to write a message to the system console terminal
TRPRT:	Non-existent memory - trap routine
$SYBEG::	Beginning of dynamic storage region
$SYTOP::	Last address in the Executive

The INITL routine resets the processor and saves the following information about the loading device:

- Unit number
- Logical block number (LBN) of load image
- Device name
- Length of load file

The INITL module sets up the basic operating parameters of the system using conditional assemblies for Memory Management, 11/70 Extended Memory Support, 11/70 Cache Parity Support, and Real Time Clock for LSI-11.

Macro Library Calls -
HWDDF$ Define hardware registers

Entry Point -
$INITL:: This is the system start up and initialization routine.

Calls $DIV, $DEACB

Input None - Processor reset

Output None - System operating parameters initialized

7.2.33 IOSUB Module

IOSUB The IOSUB module contains the following routines:

$ACHKP	Address check parameter block
$ACHKW	Address check parameter block; word aligned
$ACHK2	Address check 2-byte directive parameter block
$ACHKB	Address check; byte aligned
$ACHCK	Address check; word aligned
$ASUMR	Assign UNIBUS mapping registers
$CEFN	Convert event flag number for directive
$CEFI	Convert event flag number for I/O
$DEUMR	Deassign UNIBUS mapping registers
$DQUMR	Dequeue from UMR wait
$DVMSG	Device message output
$GTPKT	Get I/O packet from request block
$BLKCK	Check logical block
$BLKC1	Check logical block (alternate entry)
$IODON	I/O done
$IOALT	I/O done (alternate entry)
$IOFIN	I/O finish
$IOKIL	I/O kill
$LCKPR	Lock and unlock processing routine
$MPLNE	Map logical unit number for exit
$MPLUN	Map logical unit number
$MPPHY	Map to physical address
$MPPKT	Map I/O packet function
$MPUBM	Map UNIBUS to memory
$MPVBN	Map virtual block number
$RELOC	Relocate user virtual address
$RELOM	Relocate and map address
$RLCH	Release channel
$RQCH	Request channel
$SCDVT	Scan device tables
$SCDV1	Scan device tables (alternate entry)
$STMAP	Set up UNIBUS mapping address
$ECCOR	Common ECC correction code for RP04/RK06
$RELOP	Relocate UNIBUS phisical address

$CRPAS	Common register pass routine
$MUL	Integer multiply magnitude numbers
$WTUMR	Wait for change in UMR state
$DIV	Integer divide magnitude numbers

Macro Library Calls -

F11DFS	Define window and lock block offsets
HDRDF$	Define task header offsets
HWDDF$	Define hardware registers
PCBDF$	Define partition control block offsets
PKTDF$	Define I/O packet offsets
TCBDF$	Define task control block offsets

Entry Points -
$ACHKP::
$ACHKW::
$ACHK2::

Executive code calls these routines to check the address of a task specified parameter block to ensure that the block is within the task's address space and is correctly aligned. If either check fails, the routines return a directive status of 'D.RS98'.

Calls — $ACHCK, $RELOC

Input — R0 = Starting address of the block to be checked
R1 = Length of the block to be checked in bytes

Output — These routines return a directive status of 'D.RS98' to the calling routine if either check fails.

Note — Registers R0 and R3 are preserved across the call.

Entry Points -
$ACHKB::
$ACHCK::

Executive code calls these routines to check the address of a block of memory to be sure it lies within the address space of the current task.

Input — R0 = Starting address of the block to be checked
R1 = Length of the block (in bytes) to be checked

Output — C = 0 if address check succeeded.
C = 1 if address check failed.

Note — Registers R0 and R3 are preserved across the call

Entry Point -
$ASUMR

This routine assigns UNIBUS mapping registers (UMRs). It assigns a contiguous set of UMRs. For the sake of speed, the link word of each mapping assignment block points to the UMR address (2nd) word of the block, not the first word. A linked list of mapping assignment blocks represents the current state of UMR assignment. Each block contains the address of the first UMR assigned and the number of UMRs assigned times 4. The blocks are linked in the order of increasing first UMR address.

Calls — None

Input — R0 = Address of a mapping register assignment block
M.UMRN(R0) = Number of UMRs required times 4

Output — C = 0 if $ASUMR successfully assigned the UMRs. All fields of the mapping register assignment block are initialized and the block is linked into the assignment list.
C = 1 if $ASUMR could not assign the UMRs.

Note — All registers are preserved.

Entry Points -
$CEFN::
$CEFI:: — Executive code calls these routines to convert an event flag number to an event flag mask word and event flag mask address.

Calls — None

Input — R0 = Event flag number to be converted
R5 = TCB address of the task to which the event flag applies

Output — C = 0 if an event flag number was specified.
R0 = Event flag mask word
R1 = Event flag mask address

C = 1 if no event flag number was specified.
R0 = Zero
R1 = Zero

Directive status returned:
'D.RS97' if an incorrect event flag number is specified.

Note — If the $CEFI routine is called, R3 is preserved; otherwise, $CEFN adds two to R3.

Entry Point -
$DVMSG:: — This routine queues a message to the task termination notification task. The messages are related to a device failure or a checkpoint write failure occurring from the loader.

Calls — $ALOCB, $EXRGF

Input — R0 = Message number
R5 = Address of the UCB or TCB to which the message applies

Output — $DVMSG calls $ALOCB to allocate a four-word packet and stores the message number (R0) in the second word and the UCB or TCB address (R5) in the third word. $DVMSG threads the packet in the task termination notification task message queue.

Note — If the task termination notification task (SYSGEN option) is not installed, or no storage can be obtained, $DVMSG performs no function and returns to the calling routine.

Entry Point -
$GTPKT:: Device drivers call this routine to dequeue the next I/O request to be processed. If the device controller is busy, GTPKT sets the carry bit and returns to the caller. If the device controller is not busy, GTPKT tries to dequeue the next request from the controller queue. If $GTPKT cannot dequeue a request, it sets the carry bit and returns to the caller. If the $GTPKT process succeeds, $GTPKT sets the controller to busy and clears the carry bit before returning to the caller.

Calls $IOALT, $MPPKT, $EXRQP

Input R5 = Address of the UCB of the controller for which $GTPKT will get a packet.

Output C = 0 if $GTPKT successfully dequeued a packet. Also, $GTPKT returns the following contents of R1 through R5:
R1 = Address of the I/O packet
R2 = Physical unit number
R3 = Controller index
R4 = Address of the status control block
R5 = Address of the unit control block

C = 1 if the controller is busy or no request can be dequeued

Note The contents of R4 and R5 are changed by this routine.

Entry Points -
$BLKCK::
$BLKC1:: I/O device drivers call these routines to check the starting and ending logical block numbers of an I/O transfer to a file structured device. If the range of blocks is not correct, $BLKCK enters $IODON with a final status of 'IE.BLK' and then a return to the calling driver occurs at the driver's initiator entry point. If the range of blocks is correct, $BLKCK returns to the calling driver.

Input R1 = Address of the I/O packet
R5 = Address of the unit control block (UCB)

Output If the check fails, $BLKCK enters $IODON with a final status of 'IE.BLK' and $IODON returns to the calling driver at the initiator entry point.

If the check succeeds, $BLKCK returns the following contents of registers R0 through R3 to the calling driver:
R0 = Low part of logical block number
R1 = Points to I.PRM+12 (low part of user logical block number)
R2 = High part of logical block number
R3 = Address of I/O packet

Entry Point -
$DEUMR:: This routine deassigns a contiguous block of UMRs. If the mapping assignment block is not in the list, no action is taken. For the sake of assignment speed, the link word points to the UMR address (2nd) word of the assignment block.

Calls None

Input R2 = Pointer to assignment block

Output None

Note R0 and R1 are preserved.

Entry Point -
$DQUMR:: Control is transfered here to see if a driver is waiting for UMR assignemnt. $DQUMR calls the calling driver back as a co-routine. When the calling driver issues a return back to this routine, $DQUMR checks to see if any drivers are waiting for UMRs. If so, $DQUMR restores the waiting driver's context without actually de-queueing the mapping assignment block and passes control back to the original UMR assignment routine.

Input (SP) = Return address to the driver's caller

Output None

Entry Points -
$IODON::
$IOALT:: I/O device drivers call this routine at the end of an I/O request to do final processing. $IODON sets the unit and controller to idle and enters $IOFIN to finish processing.

Calls $QEMB, $DEUMR, $FORK0

Input R0 = First I/O status word
R1 = Second I/O status word. If the entry to this routine is at $IOALT, $IOALT clears R1 to signify that the second status word is zero (0).
R2 = Starting and final error retry counts if this process is the end of I/O on an error logging device
R5 = Address of the unit control block of the unit being completed
(SP) = Address of the driver's caller (for return)

Output The unit and controller are set idle.

R3 = Address of the current I/O packet

Entry Point -
$IOFIN:: This routine is called to finish I/O processing in cases where the unit and controller are not to be declared idle.

Calls $SETF, $CHKPT, $NXTSK, $QINSF, $DEPKT

Input R0 = First I/O status word
R1 - Second I/O status word
R3 = Address of the I/O request packet

Output $IOFIN:

1. Stores the final I/O status values in the I/O status block if one was specified
2. Decrements the I/O count and clears TS.RDN in case the task was blocked for I/O rundown.
3. Clears TS.CKR if it is set and initiates checkpointing of the task.
4. Queues an AST for the task if an AST service routine was specified. Otherwise, $IOFIN deallocates the I/O packet.
5. Sets the event flag.

Note $IOFIN destroys the contents of R4.

Entry Point -
$IOKIL:: This routine flushes all I/O requests for the current task from a device queue and cancels the I/O operation in progress for the current task.

Calls $IOFIN, @D.VCAN(R2) - where R2 is the address of the driver dispatch table

Input R5 = Address of the device UCB of the device for which to flush requests

Output $IOKIL calls the driver at the cancel I/O operation entry point with the arguments:
R0 = Address of the current I/O packet
R1 = Address of the TCB of the current task
R3 = Controller index
R4 = Address of the status control block
R5 = Address of the unit control block

Note $IOKIL destroys the contents of R4.

Entry Point -
$LCKPR:: This is the lock/unlock processing routine. This routine first determines if a file I/O request is to a shared file. If it is, $LCKPR determines if the request is an UNLOCK QIO or a virtual block I/O request. It then either performs the unlock QIO or the lock processing, respectively.

Calls $ALOCB

Input

The inputs for the main entry point, $LCKPR, are:
R1 = I/O packet address of the request

Unlock processing: The section of $LCKPR that performs the unlock processing contains the following relevant register contents:
R0 = Unlock error status
R1 = I/O packet address
R2 = Address of the first lock block in the lock list
R3 = Pointer to current window
R4 = Byte count of current unlock request

Lock processing: The section of $LCKPR that performs the lock processing checks for attempted lock overlaps, tries to set the new lock, and performs the implied unlock. If a new lock request for an explicit unlocker is detected that exactly matches an existing lock for that window in both starting VBN and size, the lock block is reused. The relevant registers contents for this part of $LCKPR are:
R1 = I/O packet address
R2 = Address of first lock block in lock list
R3 = Address of file window
R4 = Block count for current request

$LCKPR contains an internal routine to check for exact VBN and block count match. The inputs to this routine are:
R1 = I/O packet address
R2 = Lock block address
R4 = Byte count

The outputs of this routine are:
Z = 1 if there is an exact match
Z = 0 if there is no match
All registers are preserved.

Output

The outputs of $LCKPR are:
C = 0 if no lock processing was required

C = 1 if an unlock was performed or an error condition occured during the lock processing.
R0 = I/O status

Note

R1 is preserved.

Entry Point -
$MPLNE::
$MPLUN::

These routines validate a logical unit number (LUN) and map the LUN into a UCB pointer. If the calling routine specified an incorrect LUN, $MPLNE returns a directive status of 'D.RS96'. If the LUN is correct, $MPLNE maps it and returns the pointer, which points to the LUN and the UCB, to the calling routine.

Input

R3 = Address of the LUN
R4 = Address of the header of the current task
R5 = Address of the TCB of the current task

Output

R1 contains the address of the second LUN word in the task header.

R3 is advanced by two (2).

C = 0 if a device is assigned to the specified LUN.
R0 = Address of the UCB of the assigned device

C = 1 if no device is assigned to a specified LUN. R0 contains zero (0).

Entry Point -
$MPPHY:: This routine maps a relocation bias and displacement address to an 18-bit physical address. If the indicated device is not a non-processor request (NPR) device, $MPPHY returns the relocation bias and displacement address to the caller. Otherwise, $MPPHY converts the relocation bias and displacement address to an 18-bit physical address and returns this address to the calling routine.

Calls None

Input
R1 = Relocation bias
R2 = Displacement address
R5 = Address of the unit control block

Output
If the device is an NPR device:
R1 = High order 2 bits of physical address in bits 4 and 5
R2 = Low order 16 bits of physical address

If the device is an NPR device on an 11/70:
R1 = High order 6 bits of physical address in high byte
R2 = Low order 16 bits of physical address

If the device is not an NPR device:
R1 = Relocation bias
R2 = Displacement address

Note R0 and R3 are preserved across the call.

Entry Point -
$MPPKT:: This routine maps a read or write virtual function in an I/O packet to a read or write logical function. If the current window does not map the virtual function, MPPKT sets the C-bit and returns the partial mappping results to the calling routine. If the window completely maps the virtual function, $MPPKT stores the logical block number in the I/O packet and converts the read or write virtual function to its logical counterpart.

Calls $MPVBN, $MPPHY

Input R1 = Address of the I/O packet

Output
C = 0 if mapping was successful.
R0 = Zero (0)
I.FCN+1(R1) = IO.WLB or IO.RLB
I.PRM+10(R1) = High part of mapped LBN
I.PRM+12(R1) = Low part of mapped LBN

C = 1 if mapping failed.
R0 = Number of blocks not mapped
R2 = High part of mapped LBN
R3 = Low part of mapped LBN

Note R1 is preserved across call

Entry Point -
$MPUBM:: UNIBUS NPR device drivers call this routine to load the necessary UNIBUS map registers to enable a transfer to main memory on an 11/70 processor with extended memory.

Calls None

Input R4 = Address of device SCB
R5 = Address of device UCB

Output $MPUBM loads the necessary UNIBUS map registers to enable a transfer.

Note Register R3 is preserved across the call.

Entry Point -
$MPVBN:: This routine maps a virtual block number (VBN) to a logical block number (LBN) by using a window block that contains a set of mapping pointers.

Calls None

Input R0 = The number of consecutive bytes that must be mapped
R1 = Address of the window block
R2 = High part of VBN
R3 = Low part of VBN

Output C = 0 if $MPVBN successfully maps the VBN to the LBN. Also:
R0 = The number of unmapped blocks
R2 = High part of LBN
R3 = Low part of LBN

C = 1 if $MPVBN could not map the VBN by using the window block

Entry Point -
$RELOC:: This routine relocates a user's virtual address. $RELOC transforms a 16-bit user virtual address into a relocation bias and displacement-in-block relative to APR6.

Calls None

Input R0 = User's virtual address to be relocated

Output R1 = Relocation bias to be loaded into PAR6
R2 = Displacement-in-block plus 140000 (PAR6 bias)

Note R0 and R3 are preserved across the call.

Entry Point -
$RELOM:: This routine transforms a 16-bit user virtual address into a relocation bias and displacement-in-block relative to APR6 and loads these values for access by the caller.

Calls $RELOC

Input R0 = User's virtual address to be relocated

Output R0 = Displacement-in-block

$RELOM loads KISAR6 with the relocation bias

Note R3 is preserved across the call.

Entry Point -

$RLCH:: This routine releases a channel. It sets the channel status to idle and tries to dequeue the next driver waiting to use the channel. If no driver is waiting, $RLCH returns to the calling routine. Otherwise, $RLCH dequeues the driver, sets the channel status to busy, calls the driver, and returns to the calling routine.

Calls $QRMVF

Input R5 = Address of the unit control block

Output $RLCH sets the channel status to idle and tries to dequeue the next driver waiting to use the channel.

R0, R1, and R2 are preserved across the call.

Note $RLCH destroys the contents of R4.

Entry Point -

$RQCH:: This routine requests exclusive use of a channel. If the channel is busy, $RQCH threads the calling driver into the channel wait queue and returns to the routine that called the driver. If the channel is not busy, $RQCH sets the channel status to busy and returns to the calling driver.

Calls $QINSF

Input R4 = Address of status control block
R5 = Address of unit control block
(SP) = Return address of calling routine
2(SP) = Return address of the routine that called the calling routine

Output The calling driver is threaded into the channel wait queue.

Entry Point -

$SCDVT::
$SCDV1:: This co-routine scans device tables for a calling routine. For each UCB found, this co-routine calls the calling routine and returns the UCB, DCB, and SCB addresses to it.

Calls @(SP)+ (the calling routine)

Input R3 = List pointer (if entry is at $SCDV1)

Output C = 0 if the next device table entry is being returned
R3 = Address of the device control block (DCB)
R4 = Address of the status control block (SCB)
R5 = Address of the unit control block (UCB)

C = 1 if no more entries exist in the device tables

Entry Point -

$STMAP:: UNIBUS NPR device drivers call this routine to set up the UNIBUS mapping address. $STMAP first assigns the UNIBUS mapping registers (UMRs). If the UMRs cannot be allocated, $STMAP places the driver's mapping assignment block in a wait queue and returns to the driver's caller. The assignement block will be dequeued eventually when the UMRs are available and the driver will be remapped and returned to with R1 through R5 preserved and the normal outputs of this routine. $STMAP stores the driver's context in the assignment block and the fork block while it is blocked and in the wait queue. Once $STMAP places the driver's mapping assignment block in the UMR wait queue, it is not removed from the queue until the UMRs are successfully assigned. This strategy assures that waiting drivers are serviced FIFO and that drivers with large requests for UMRs will not wait indefinitely.

Calls $ASUMR, $WTUMR

Input R4 = Address of the device SCB
R5 = Address of the device UCB
(SP) = Return to driver's caller

Output $STMAP sets up UNIBUS map addresses in the device UCB and moves the actual physical address to the SCB.

Note Registers R1, R2, and R3 are preserved across the call.

Entry Point -

$ECCOR:: This routine contains common error correction code (ECC) for the RP04 and RK06. $ECCOR applies the ECC correction algorithm and determines if offset recovery is required (if supported).

Calls $RELOP

Input R1 = Contents of error register
R2 = Control status register (CSR) address
R3 = Address of first ECC register
R5 = UCB address

Output R3 = Controller index

C = 0 if offset recovery is not required
R0 = IS.SUC&377
R1 = bytes actually transfered
R2, R4, and R5 are unmodified
U.BUF and U.BUF+2 are updated by a call to $RELOP

C = 1 if offset recovery is required
R0 = Number of good bytes transfered
R1 - R5 are unmodified

Entry Point -
$RELOP:: This routine relocates a UNIBUS physical address to a KISAR6 bias and displacement.

Calls None

Input R0 = byte offset from address in U.BUF+1 and U.BUF+2
R5 = UCB address
U.BUF+1(R5) = High order bits of physical address
U.BUF+2(R5) = Low order bits of physical address

Output KISAR6 = Calculated bias (mapped system)
R1 = Real address or displacement

Entry Point -
$CRPAS:: This is the common register pass routine. $CRPAS passes the contents of the device registers back to the diagnostic task. This routine passes all registers in the order in which they appear on the UNIBUS. $CRPAS uses the error logging entries, S.ROFF and S.RCNT, in the status control block. To use this routine, error logging must be enabled.

Calls None

Input R1 = I/O packet address
R2 = Control status register address
R4 = Status control block address

Output None

Note R0 and R2 are preserved.

Entry Point -
$MUL: This is the integer multiply routine.

Calls None

Input R0 = Multiplier
R1 = Multiplicand

Output $MUL returns a doubleword result in R0 and R1. The high part is in R0 and the low part is in R1.

Note Registers R2, R3, R4, and R5 are preserved across the call.

Entry Point -
$DIV:: This is the integer divide routine.

Calls None

Input R0 = Dividend
R1 = Divisor

Output $DIV returns the quotient in R0 and the remainder in R1.

Note Registers R2, R3, R4, and R5 are preserved across the call.

Entry Point -

$WTUMR:: This routine waits for a change in the UMR state. It stores R1 through R4 and the return PC in the mapping assignment block and queues the block in the UMR wait queue for a subsequent recall to the caller when the state of the UMRs changes. It is possible for the mapping assignment block to already be in the wait queue. If it is, it can be at the head of the queue only.

Calls None

Input R0 = Pointer to UMR assignment block

Output $WTUMR queues the assignment block in the UMR wait queue. $WTUMR returns to the caller at system state with R1 through R4 preserved.

7.2.34 LOADR Module

LOADR The LOADR module is a task that loads and checkpoints all nonresident tasks.

Macro Library Calls -

ABODF$	Define task abort codes
HDRDF$	Define task header offsets
HWDDF$	Define hardware registers
PCBDF$	Define partition control block offsets
TCBDF$	Define task control block offsets

Entry Point -

$LOADR:: This task:

- Reads a nonresident task into memory and initializes it for execution.
- Reads a previously checkpointed task back into memory and restarts its execution.
- Writes a checkpoint image of a running task and frees its partition.

$LOADR gets its input from a loader queue by a call to $QRMVF. The loader queue contains a priority ordered list of task TCBs. $LOADR removes the highest priority (the first) TCB from the queue and processes it. $LOADR processes the TCBs in the queue in priority sequence.

Calls $QRMVF, $STPCT, $DEACB, $QINSP, $RLPAR, $DVMSG, $ALOCB, $TKWSE, $MAPTK, $CRATT, $SWSTK, $BILDS, $CHKPT, $DASTT, $ACTTK, $ABTSK

7.2.35 LOWCR Module

LOWCR The LOWCR module contains low core pointers, trap and interrupt vectors, and the Executive stack area. This module contains one executable statement - a JUMP to the panic dump routine ($PANIC). The other statements define the addresses of trap processing modules and low core pointers.

This file must be the first in the task builder command file because it occupies locations starting at real location zero.

The following low core pointers are contained in this module:

- Address of directive status
- Directive status word
- FCS impure area pointer
- FORTRAN impure area pointer
- Overlay run time system impure area pointer

The system stack area contains a minimum of 40. words.

Calls None

Input None

Output None

7.2.36 PARTY Module

PARTY Execution of this module occurs because of a memory parity error interrupt. Any error that occurs on the system stack or in the Executive is a fatal error. A fatal error halts the system and the message "***EXEC PARITY ERROR***" is printed. Otherwise, the task occupying the memory in which the error occurred is aborted and locked into memory to prevent that portion of memory from being used again. A message is printed to indicate that the task was aborted.

Macro Library Calls -

ABODF$	Define task abort codes
HWDDF$	Define hardware registers
PCBDF$	Define partition control block offsets
TCBDF$	Define task control block offsets

Entry Point -

PARER: Memory parity error interrupt processing module

Calls $ALEMB, $QEMB, $ABTSK

Input None

Output None

Note

The error message block (EMB) format (for error logging) for cache parity is:
WD. 00 - Link word
WD. 01 - Size = 37.*2
WD. 02 - Processor type/entry code = 002
WD. 03 - Minute/second
WD. 04 - Day/hour
WD. 05 - Year/month
WD. 06 - Error sequence number
WD. 07 - Trap PC
WD. 08 - Trap PS
WD. 09 - First word of task name
WD. 10 - Second word of task name
WD. 11 - First word of partition name
WD. 12 - Second word of partition name
WD. 13 - Partition base address
WD. 14 - Partition size
WD. 15 to WD. 30 - Memory parity control status registers (CSRs)
WD. 31 to WD. 36 - Cache parity CSRs

7.2.37 PLSUB Module

PLSUB

The PLSUB module contains the following program logical address space subroutines:

$SRNAM	Search for a named partition
$CKACC	Check desired access
$CRATT	Create attachment descriptor
$SRATT	Search for attachment descriptor
$SRWND	Search for specified address window
$UNMAP	Unmap address window

Macro Library Calls -

HDRDF$	Define task header offsets
HWDDF$	Define hardware registers
PCBDF$	Define partition control block offsets
TCBDF$	Define task control block offsets

Entry Point -

$SRNAM:: This routine searches for a named partition and returns a success or failure indication and a pointer to the PCB if it finds the partition.

Calls None

Input R3 = Pointer to the double-word RAD50 name

Output C = 0 if SRNAM finds the named partition.
R2 = The PCB address

C = 1 if SRNAM does not find the name

Entry Point -
$CKACC:: This routine checks if the desired access of a task to a region is allowed. The bits in the protection word are arranged in the following order:

```
WORLD,GROUP,OWNER,SYSTEM
15                     0
```

The bits within each category in the protection word are arranged in the follwing order:

```
DELETE  EXTEND  WRITE  READ
3       2       1      0
```

If a bit is set, the corresponding access is not allowed.

Calls None

Input R0 = Desired access mask in the low four bits
R1 = Current UIC of the task
R2 = PCB address of the region

Output CKACC changes the contents of R0 and R1.

CKACC returns a directive status of 'D.RS16' to the calling routine if access to the region is not possible.

Entry Point -
$CRATT:: This routine creates an attachment descriptor block and inserts it in the necessary queues.

Calls $ALOCB, $QINSF, $QINSP

Input R2 = PCB address of the region being attached to
R4 = Access code
R5 = TCB address of attaching task

Output C = 0 if CRATT successfully completes execution.
R1 = Address of the attachment descriptor block
CRATT changes the contents of R0

C = 1 if an attachment descriptor could not be allocated.
CRATT modifies R0 and R1.

Entry Point -
$SRATT:: This routine verifies that a valid region ID was passed in a PLAS directive by searching for the corresponding attachment descriptor in the task control block's attachment queue.

Calls None

Input R3 = Address of the region ID to be verified (or 0 for task region)
R5 = TCB address of the current task

Output R5 = Address of the attachment descriptor

SRATT returns a directive status of D.RS86 if the attachment descriptor cannot be found.

Entry Point -
$SRWND:: This routine verifies that the specified address window ID corresponds to a valid established address window.

Calls None

Input R3 = Address of the address window ID
R4 = Address of the current task header

Output R4 = Pointer to the specified window block

SRWND chages the contents of R0

SRWND returns a directive status of D.RS87 if the specified address window is invalid.

Entry Point -
$UNMAP:: This routine searches for and conditionally unmaps the specified address window.

Calls None

Input R4 = Address of the window to be unmapped

Output UNMAP modifies R0

C = 1 if UNMAP sucessfully unmapped the address window

7.2.38 POWER Module

POWER This is the powerfail recovery module. If power fails, the POWER module saves the stack pointer, hardware registers R0 - R5, UNIBUS mapping registers, memory management registers, floating point status, mode, and registers, and the program interrupt request.

When power comes back on, POWER restores all the registers, forces a schedule request for the null task, increments the powerfail indicator, and executes an RTI instruction.

The dispatcher then calls the $POWER routine in this module to restart the system.

The POWER module contains the following labels and routines:

PWBTM: This is the label of register storage area
PWSTK: This is the label of the stack pointer storage area
PWVCT: Defines the powerfail vector
PDOWN: This routine is entered when a power failure interrupt occurs. This routine saves all volatile machine registers, switches the power fail vector to the power up routine (PUP:), and halts the processor to await the power up interruption.

PUP: This routine is entered when a power up interrupt occurs. It restores all volatile machine registers, forces a schedule request for the NULL task, increments the powerfail indicator, and executes an RTI instruction. The dispatcher then calls the power recovery routine ($POWER::), which is in this module, to re-instate system processing.

$POWER::Restart system processing

Macro Library Calls -

EMBDF$	Define error message block offsets
HDRDF$	Define task header offsets
HWDDF$	Define hardware registers
PCBDF$	Define partition control block offsets
TCBDF$	Define task control block offsets

Entry Point -

$POWER:: The dispatcher calls the POWER routine after the PUP routine restores the volatile registers. The POWER routine clears the power failure indicator and declares a significant event. If the KW11-Y is supported, POWER clears the clock error flags and energizes the output relay. POWER then enables parity error interrupts, clears the 11/70 parity control register, clears the memory error register, declares power failure ASTs for task that are active and in core, and performs a power failure recovery for all active devices.

Calls $DRDSE, $ALEMB, $QEMB, $DASTT, @D.VPWF(R2)-where R2 is the address of the device dispatch table

Input $POWER calls the driver at the powerfail entry point with the following arguments:
R3 = Controller index
R4 = Address of the status control block
R5 = Address of the unit control block

Output None

7.2.39 QUEUE Module

QUEUE The QUEUE module contains the following queue manipulation routines:

$CLINS::	Clock queue insertion
$CLRMV::	Clock queue removal
$QINSF::	Queue insertion at end of list
$QINSP::	Queue insertion by priority
$QMCRL::	Queue MCR command line
$QRMVF::	Queue removal from front of list
$QRMVT::	Queue removal by TCB address

Macro Library Calls -

CLKDF$	Define clock queue control block offsets
TCBDF$	Define task control block offsets

Entry Point -
$CLINS:: This routine makes an entry in the clock queue. The routine inserts the entry in a way that allows the clock queue to be ordered in ascending time sequence. Thus, the entries at the top (or front) of the queue are the most immminent.

Calls None

Input R0 = Address of the clock queue entry core block
R1 = High order half of delta time
R2 = Low order half of delta time
R4 = Request type
R5 = Address of requesting TCB or request identifier

Output The clock queue entry is inserted in the clock queue according to the time that it will become due.

Entry Point -
$CLRMV:: This routine removes all entries for a specified TCB address and request type from the clock queue.

Calls $DECLK

Input R4 = Request type
R5 = Address of requesting TCB or system subroutine

Output CLRMV removes all entries for the specified TCB address and request type from the clock queue. If the request type is not 'C.SYST', CLRMV releases the clock queue entry core block.

Entry Point -
$QINSF:: This routine makes an entry in a first-in first-out list. It links the entry to the end of the list. This routine shares common code with the $QINSP routine.

Calls None

Input R0 = Address of the two-word listhead
R1 = Address of the entry to be inserted

Output The entry is linked to the end of the queue.

R0 and R1 are preserved across the call.

Entry Point -
$QINSP:: This routine inserts an entry in a priority ordered list. QINSP searches the list until it finds an entry that has a lower priority or it finds the end of the list. The new entry is linked into the list at whichever of these two points is appropriate.

Calls None

Input R0 = Address of the two-word listhead
R1 = Address of the entry to be inserted

Output QINSP links the entry into the list in priority order.

R0 and R1 are preserved across the call.

Entry Point -
$QMCRL:: This routine queues a command line for MCR.

Calls $EXRQF

Input R1 = Address of the command line control block

Output Via the call to EXRQF, the command line is inserted into the MCR command line list and MCR is requested to run.

Entry Point -
$QRMVF:: This routine removes the next (front) entry from a list. The list organization may be either FIFO or by priority.

Calls None

Input R0 = Address of the two-word listhead

Output C = 0 if QRMVF removes the next entry from the list R1 = Address of the entry removed

C = 1 if there are no entries in the list R0 is preserved across the call

Entry Point -
$QRMVT:: This routine removes the next entry from a list that matches a specified TCB address. The list organization may be either in FIFO or in priority order.

Calls None

Input R0 = Address of the two-word listhead
R1 = Address of the TCB for which QRMVT searches

Output C = 0 if QRMVT removes a matching entry from the list
R1 = Address of the entry removed

C = 1 if there is no entry in the list that matches the TCB address

R0 is preserved across the call

7.2.40 REQSB Module

REQSB This module contains the following task control routines:

$ABCTK	Abort current task
$ABTSK	Abort task
$BILDS	Build stack and initialize header
$ACTTK	Put task in active task list
$SETCR	Set conditional schedule request
$SETRQ	Set schedule request
$SETRT	Set schedule request for current task
$SETF	Set event flag
$SETM	Set event flag
$DASTT	Declare AST trap
$DQAC	Dequeue AST block queued by $QASTC (called from SYSXT only)
$QASTC	Queue AST to task
$QASTT	Queue AST to task
$SRSTD	Search system task directory

$ACTRM	Remove task from the active task list
$STPCT	Stop current task
$STPTK	Stop task
$RLPAR	Release task partition
$RLPR1	Release partition
$NXTSK	Assign next task to partition
$FNDSP	Find space in partition control block (PCB) list
$TSTCP	Test if checkpoint should be initiated
$ICHKP	Initiate checkpoint
$CHKPT	Checkpoint task
$LOADT	Put task in loader queue
$EXRQP	Executive request with queue insert by priority
$EXRQF	Executive request with queue insert FIFO
$EXRQN	Executive request with no queue insertion
$TSKRT	Task execution request (default UCB)
$TSKRQ	Task execution request (UCB specified)
$TSKRP	Task execution request (default UIC specified)
$UISET	Establish default UIC and current UIC
$MAPTK	Map task address window

Macro Library Calls -

HDRDF$	Define task header offsets
ITBDF$	Define interrupt transfer block offsets
PCBDF$	Define partition control block offsets
TCBDF$	Define task control block offsets

Entry Point -

$ABCTK:: Abort current task

$ABTSK:: Abort task

These routines mark a task to be aborted and they force a task's exit. They store the abort reason and the current outstanding I/O count in the first task event flag word. If the entry occurs at $ABTSK, R1 contains the address of the TCB of the task to be aborted. If the entry occurs at $ABCTK, the TCB address of the task to be aborted is moved into R1. Otherwise, the remainder of the two routines have common code.

Calls $SETCR, $NXTSK

Input R0 = Reason for the abort
R1 = Address of the TCB of the task to be aborted when entry occurs at $ABTSK only.

Output These routines mark the task to be aborted and set a conditional schedule request.

Entry Point -

$BILDS:: This routine sets up the task stack and initializes the header. This routine is called prior to placing a task into contention for the processor. This occurs when an execution request is made for a task that is fixed in memory or when a disk resident task has finished loading.

Calls None

Input R0 = Address of the TCB of the task to initialize

Output $BILDS::
1 - Clears the task local event flags 1. through 32.
2 - Sets up the current UIC in the header
3 - Sets up task context (PC, PS, stack pointer) to cause the task to start execution at its entry
4 - Conditionally requests redispatching of the processor

Entry Point -
$ACTTK:: This routine puts an active task in the active task list.

Calls None

Input R0 = Address of the TCB of the task to be put in the active task list.

Output ACTTK merges the specified task into the active task list by priority.

R3 is preserved across the call.

Entry Point -
$SETCR Set conditional schedule request
$SETRQ Set schedule request
$SETRT Set schedule request for current task
These routines force redispatching of the processor from a specified position in the active task list. If a previous request was set, redispatching starts at whichever request has the highest priority.

These routines share common code.

Calls $DRDSE

Input If entry occurs at $SETRT:
R5 = Address of the TCB of the current task

If entry occurs at $SETRQ or $SETCR:
R0 = Address of the TCB at which dispatching is to start

Output These routines set a schedule request that forces a redispatching of the processor when a system exit is executed.

R2 and R3 are preserved across the call.

R0 is preserved across the call if entry occurs at $SETRQ or $SETCR.

Entry Point -
$SETF $SETM These routines set an event flag and do the required rescheduling.

Calls CEFI, $DRDSE

Input R0 = Event flag number ($SETF) or event flag mask ($SETM) only)
R1 = Event flag word address ($SETM only) set
R5 = TCB address for which flag is being set

Output R0 = TCB address of the task whose flag was set

Note R3 is preserved

Entry Point -
$DASTT:: This routine declares a non-I/O related ast trap. It examines the header of the specified task to determine if the specified AST is enabled. If it is enabled, the AST is declared.

Calls None

Input R4 = Offset into the task header to the AST control block address
R5 = Address of the TCB of the task for which the AST is to be declared

Output C = 0 if DASTT succeeded in setting up the task for the AST and declaring the AST
R1 = Address of the AST control block

C = 1 if the task is not setup for the specified AST

Note DASTT alters the contents of R4 during execution

Entry Point -
$DQAC This routine dequeues an AST block that was queued by $QASTC

Calls None

Input R0 = Pointer to AST block

Output A.CBL is set to one (1) to indicate that the AST block is free (not in AST queue)

Entry Point -
$QASTC Queues an AST to a task. This routine is a variant of $QASTT used by a task ISR which was specified in a CINT$ directive.

Calls None

Input R5 = Pointer to fork block in ITB

Output C = 0 if the AST is queued.
C = 1 if the AST address was not specified in the CINT$ call.

Note If the AST block is already queued for the task, $QASTC takes no action and returns C = 0.

$QASTC alters R0, R1, R2, and R3.

Entry Point -
$QASTT:: This routine queues an AST to a task and ensures that the task will be scheduled and reconsidered for eligibility in the partition.

Calls $QINSF, $SETCR, $NXTSK

Input R0 = TCB address of the task to receive the AST
R1 = Address of the AST control block to be used

Output C = 0 if QASTT successfully completed execution

R1 is preserved across the call

Entry Point -
$SRSTD:: This routine searches the task directory for a task of the specified name.

Calls None

Input R3 = Address of the task name for which to search

Output C = 0 if SRSTD finds the task.
R0 = Address of the TCB

C = 1 if SRSTD did not find the specified task

R1, R2, and R3 are preserved across the call

Entry Point -
$ACTRM:: This routine removes a task (its TCB) from the active task list.

Calls None

Input R0 = Address of the TCB to be removed

Output C = 0 if ACTRM succeeds in removing a matching entry from the list

C = 1 if there is no entry in the list that matches the TCB address

R0 is preserved across the call

Entry Point -
$STPCT:: Stop the current task
$STPTK:: Stop the task
These routines stop a task and reallocate the task's partition.

Calls $SETCR

Input R0 = The TCB of the task to be stopped (if entry occurs at $STPTK)

Output None

Entry Point -
$RLPAR:: Release task partition
$RLPR1:: Release partition
This routine releases a partition owned by a task and assigns the partition to the next highest priority task waiting to occupy the partition.

Calls None

Input R0 = Address of the TCB of the owner task (if entry occurs at $RLPAR)
R1 = Address of subpartition PCB to release (if entry occurs at $RLPR1)
R3 = Address of main partition PCB (if entry occurs at $RLPR1)

Output The partition is released and assigned to the next highest priority task waiting to occupy the partition.

Entry Point -
$NXTSK:: This routine assigns a partition to the highest priority task waiting to occupy the partition.

Calls $QRMVT, $LOADT, $TSTCP, $ICHKP, $FNDSP

Input R0 = Address of the PCB of the partition to be assigned

Output Five outputs are possible:

1 - The partition is not currently busy and a task is waiting to occupy the partition. NXTSK assigns the partition to the waiting task and places a request in the loader queue to load the task.
2 - The partition is currently occupied by a task that is either of higher priority than all waiting tasks or is not checkpointable. In this case, the partition cannot be assigned to another task.
3 - The partition is currently occupied by a lower priority checkpointable task. NXTSK places a request in the loader queue to checkpoint the owner task.
4 - The highest priority task waiting to occupy the partition requires the main partition while the main partition is occupied by one or more tasks that are either of higher priority or are not checkpointable. In this case the partition cannot be assigned.
5 - The highest priority task that is waiting to occupy the partition requires the main partition. One or more tasks of lower priority that are checkpointable currently occupy the main partition. NXTSK places a request in the loader queue to checkpoint each task.

Entry Point -
$FNDSP:: This routine finds space within a dynamically allocated PCB list. The list represents the allocation state of a system controlled partition or dynamic checkpoint file.

Calls None

Input R4 = Address of the PCB for which to find space
R5 = Address of the main PCB in the list where space is to be found

Output C = 0 if FNDSP successfully allocated space and linked the sub-PCB into the allocation list.

C = 1 if FNDSP was unsuccessful

Note FNDSP changes the contents of R0, R1, and R2.

Entry Point -
$TSTCP:: This routine checks the priority of the requested task to determine if the owner task should be checkpointed.

Calls None

Input R1 = Address of the TCB of the owner task
R4 = Address of the TCB of the requested task

Output C = 0 if checkpoint should occur

C = 1 if checkpoint should not occur

Entry Point -
$ICHKP:: This routine starts the process of checkpointing a task. The $CHKPT routine does the actual checkpointing.

Calls $SETCR

Input R1 = Address of the TCB of the task to be checkpointed

Output None

Entry Point -
$CHKPT:: This routine checkpoints a task.

Calls $ALOCB, $DEACB, $FNDSP

Input R1 = Address of the TCB of the task to be checkpointed.

Output $CHKPT:
1 - Sets the checkpoint flag in the task status word
2 - Places the task in the loader queue
3 - Requests the loader to checkpoint the task

Entry Point -
$LOADT:: This routine puts a task in the loader queue for an initial load or a checkpoint operation. This routine contains one instruction only, which moves the address of the TCB of the loader into R0.

Calls None

Input R1 = Address of the task control block

Output None

Note This routine shares code with the $EXRQP, $EXRQF, and $EXRQN routines.

Entry Point -
$EXRQP:: Executive request with queue insert by priority
$EXRQF:: Executive request with queue insert FIFO
$EXRQN:: Executive request with no queue insertion
The Executive uses these routines when requesting all tasks.

Calls $SETCR

Input R0 = TCB address of the task to be requested
R1 = Address of the packet to be queued to the task (if entry occurs at $EXRQP or $EXRQF)

Output C = 0 if the request successfully completed execution.
C = 1 if the task was not requested and,
Z = 0 if the PCB could not be allocated
Z = 1 if the task is active, being removed, or being fixed

Entry Point -
$TSKRT:: Task request (default UCB)
$TSKRQ:: Task request (specify UCB)
$TSKRP:: Task request (specify default UIC)
These routines request the execution of a task.

Calls $ALOCB, $QINSP, $NXTSK, $BILDS

Input R0 = Address of the TCB of the task to be requested
R1 = Request UIC
R2 = UCB address if entry at $TSKRQ
R3 = Default UIC if entry at $TSKRP

Output C = 0 if these routines successfully request the execution of the task

C = 1 if the task was not requested
Z = 1 if the task is active or is being fixed in memory
Z = 0 if the PCB could not be allocated

Note These routines share common code.

Entry Point -
$UISET:: This routine establishes the default and current UIC for requested tasks in multi-user systems.

Calls None

Input R1 = Request UIC
R2 = Address of second status word of the current task
R4 = Address of the header of the current task

Output C = 0 if UISET establishes the UIC
R1 = Current UIC
R3 = Default UIC

C = 1 if a nonprivileged task is trying to change the UIC

Entry Point -
$MAPTK:: This routine maps the first window block in a task's header in a mapped system from the task's TCB and PCB.

Calls None

Input R1 = Pointer to the number of window blocks in the task header
R5 = Address of the task control block (TCB) for the task

Output R1 = Address of the last PDR image in the task header

Note MAPTK modifies the contents of R2.

7.2.41 SSTSR Module

SSTSR — The SSTSR module contains the following synchronous system trap service routines:

$EMSST::	Non-RSX EMT/TRAP instruction processing
$FLTRP::	Floating-point exception processing (11/40)
$FLTRP::	Floating-point exception processing (11/45)
$FPINT::	Programmed interrupt request entry point
$ILINS::	Illegal or reserved instruction processing
$IOTRP::	IOT instruction processing
$SGFLT::	Segment fault processing
$TRACE::	Trace (T-bit) or break point instruction (BPT) trap processing
$TRP04::	Trap at 4 (odd address, non-existent memory, etc.) processing
$SSTXT::	Common SST exit routine

The SSTSR module contains the following local data:

1 - Floating point exception vector -
```
.WORD $FLTRP
.WORD PR7
.WORD $FPINT
.WORD PR7
```

2 - Floating point status and fork block
```
FLSTS: .BLKW 2
FLFRK: .BLKW 2
```

3 - Segment fault vector -
```
.WORD $SGFLT
.WORD PR7
```

Macro Library Calls -

ABODF$	Define task abort codes
HDRDF$	Define task header offsets
HWDDF$	Define harware registers
PKTDF$	Define I/O packet offsets

Entry Point -

$EMSST:: The directive dispatcher (DRDSP) transfers control to this routine when the system executes a non-RSX EMT or TRAP instruction. The machine state was saved before entry into this routine occurs. EMSST sets up the EMT/TRAP code (low byte of the instruction) to be passed to the user and transfers control to the SST exit routine.

Calls: None

Input: R5 = Address of the EMT/TRAP instruction

Output
04(SP) = EMT/TRAP code multiplied by 2
02(SP) = SST code (SCEMT=EMT, SCTRP=TRAP)
00(SP) = Number of bytes to be transfered to the user stack (6)

Entry Point -

$FLTRP:: The system traps to this routine when an 11/40 floating point exception occurs. FLTRP saves the current machine state and transfers control to the SST exit routine.

Calls: None

Input None

Output 02(SP) = SST code (SCFLT)
00(SP) = number of bytes to be transfered to user stack (4)

Entry Points -

$FLTRP:: The system traps to this entry point when an 11/45 floating point exception occurs. FLTRP saves the floating point exception and address registers and posts a programmed interrupt request at priority level 1.

$FPINT:: Control returns to this entry point when the programmed interrupt request, which was issued by FLTRP, is processed by the system.

Calls $INTSV, $FORK0, $DASTT

Input The floating point exception register contains the reason for the fault and the floating point address register contains the address of the instruction that caused the fault.

Output The outputs are the saved floating point exception and address registers and the posted programmed interrupt request.

Entry Point -

$ILINS:: A trap occurs to this routine when the system tries to execute an illegal or reserved instruction. This routine saves the current machine state and transfers control to the SST exit routine.

Calls None

Input None

Output 02(SP) = SST code (SCILI)
00(SP) = number of bytes to be transferred to the user stack (4)

Note This routine shares common code with the $TRP04 routine.

Entry Point -

$IOTRP:: The system traps to this routine when it executes an IOT instruction. If the stack depth is not +1, this routine crashes the system via a jump to $CRASH. If the stack depth is +1, this routine saves the current machine state and transfers control to the SST exit routine.

Calls None

Input None

Output The following arguments are set up on the current stack:
02(SP) = SST code (SCIOT)
00(SP) = Number of bytes to be transfered to the user stack (4)

Note This routine shares common code with the $TRP04 routine.

Entry Point -

$SGFLT:: The system traps to this routine when a segment fault occurs. SGFLT saves the current machine state, sets up SR0 through SR2 to be passed to the user, and transfers control to the SST exit routine.

Calls None

Input None

Output 10(SP) = Contents of SR0
06(SP) = Contents of SR2
04(SP) = Contents of SR1
02(SP) = SST code (SCSGF)
00(SP) = Number of bytes to be transfered to the user stack (10)

Entry Point -

$TRACE:: The system traps to this routine when a trace trap bit (T-bit) occurs or when the system executes a breakpoint trap instruction. This routine saves the current machine state and transfers control to the SST exit routine.

Calls None

Input None

Output 02(SP) = SST code (SCBPT)
00(SP) = Number of bytes to be transfered to the user stack (4)

Note This routine shares common code with the $TRP04 routine.

Entry Point -

$TRP04:: The system traps to this routine when a trap at 4 occurs. If a stack violation occurred (stack pointer=<400), this routine crashes the system. Otherwise, this routine saves the current system state and transfers control to the sst exit routine (SSTXT).

Calls None

Input None

Output 02(SP) = SST code (SCOAD)
00(SP) = Number of bytes to be transferred to the user stack (4)

Note This routine shares common code with the exit routine (SSTXT).

Entry Point -

$SSTXT:: This routine gets control to affect a synchronous system trap (SST). If the current system stack depth is not zero, this routine crashes the system. If the stack depth is zero, the routine tries to execute an SST for the current task. If the task does not have the appropriate SST vector entry or this routine cannot push the SST parameters onto the stack, this routine aborts the task. Otherwise, this routine sets up the SST and executes a directive exit.

Calls $ACHCK, $RELOM, $ABCTK

Input

For a mapped system:
24(SP) = PS word saved by SST trap
22(SP) = PC word saved by SST trap
20(SP) = Saved R5
16(SP) = Saved R4
14(SP) = Saved R3
12(SP) = Saved R2
10(SP) = Saved R1
06(SP) = Saved R0
04(SP) = SST parameter (zero or more parameters may be specified)
02(SP) = SST code
00(SP) = Number of bytes to be transferred to the the user stack

For a real memory system:
14(SP) = Saved R3
12(SP) = Saved R2
10(SP) = Saved R1
06(SP) = Saved R0
04(SP) = SST parameter (zero or more parameters may be specified)
02(SP) = SST code
00(SP) = Number of bytes to be transfered to the user stack

Output

The routine executes the specified SST for the current task.

7.2.42 SYSCM Module

SYSCM

The SYSCM module defines common data areas that are used by the system for storage of system data and pointers. The module does not contain executable code; it only defines system data areas. The global areas defined in this module are shown below. The null task control block areas in SYSCM are unique for this TCB. Remaining areas, as noted, are in the true system common area

For the null task control block: (This TCB terminates the system and active task lists. It must have a priority of zero and always be blocked.)

SHEADR::	T.LNK - Pointer to current task header
$CURPR::	T.IOC - Current task priority
$COMEF::	T.TCB, T.NAM - Common event flags
$SYSID::	T.NAM+2, T.RCVL - System identification
$TKNPT::	T.RCVL+2 - Pointer to TKTN TCB
$SHFPT::	T.ASTL - Pointer to shuffler TCB
$CKCNT::	T.ASTL+2 - Address of clock count register
$CKCSR::	T.EFLG - Address of clock control status register
SCKLDC::	T.EFLG+2 - Clock load count
$SYUIC::	T.UCB - System UIC (54,1 for mapped system, 50,1 for unmapped system)
$EXSIZ::	T.STAT - Address of last byte in Executive
$PWRFL::	T.ST2 - Powerfail recovery request flag
$SIGFL::	T.ST3 - Task waiting for significant event
$LOGHD::	T.NRPC - Logical device assignment list
$MCRCB::	T.LBN+1 - MCR command block address
$LSTLK::	T.LDV - Lock word (TCB address of owner)
$CRAVL::	T.MXSZ - Active task list listhead

MODULE DESCRIPTIONS

(The labels that follow are in the system common area.)

Pointers:

$ACTHD::	Active task list listhead
$ABTIM::	Absolute time counter
$TKTCB::	Current task TCB
$RQSCH::	Schedule request TCB address
$STKPD::	Stack depth indicator
$DEVHD::	First device control block
$MCRPT::	MCR TCB
$ERRPT::	Error logger TCB
$CFLPT::	First checkpoint file PCB
$INTCT::	Clock interrupt ticks count
$FRKHD::	Fork queue listhead
$FMASK::	System feature mask
$PARPT::	Parity address vector table
$CLKHD::	Clock queue
$COPT::	Command output UCB
$PARHD::	Partition list
$LDRPT::	Loader TCB
$TSKHD::	System task directory

Idle pattern:

$IDLCT::	Idle pattern count byte
$IDLFL::	Idle pattern flag byte
$IDLPT::	Idle pattern word

Days per month table:

$DYPMN::	February....January

Bit mask table:

$BTMSK::	Bit mask table

Online error logging data base:

$ERRHD::	Error logging message queue listhead
$ERRLM::	Limit on resident error logging data
$ERRSQ::	Universal error sequence number
$ERRSV::	Pointer to error file indentification
$ERRSZ::	Resident bytes of error logging data
$IOABM::	Devive I/O active bitmap

System bootstrap and save configuration vector:

$SYSIZ::	Size of memory in 32W blocks

Time Limit Parameters:

$TKPS::	Ticks per second

Current time vector:

$TTNS::	Tick of second

LIFO send and I/O preallocation list pointer and parameters:

$PKAVL::	Pointer to first packet in list
$PKNUM::	Number of packets currently in list

$PKMAX:: Maximum number allowed in list

Global task size limit for extend task directive:

$MXEXT:: Initialize to no limit (or reference label)

UMR allocation listhead and wait queue listhead:

$UMRHD:: Mapping assignment block listhead

$UMRWT:: UMR wait queue listhead

Macro Library Calls -

HDRDF$	Define task header offsets
HWDDF$	Define hardware registers
TCBDF$	Define task control block offsets

Entry Point - None

Input - None

7.2.43 SYSDF Module

SYSDF The SYSDF module globally defines the following:
V$$CTR, defining the highest vector address
S$$YDF, causing offset definitions from prefix files to be listed
ITBDF$, defining ITB offsets and length
PCBDF$, defining PCB offsets and length
SCBDF$, defining SCB offsets
TCBDF$, defining TCB length

The following directive status codes:
D.RS1==-1. Insufficient dynamic core available to satisfy request
D.RS2==-2. Specified task not installed in the system
D.RS5==-5. Unassigned LUN
D.RS6==-6. Driver not loaded
D.RS7==-7. Task not active
D.RS8==-8. Task not suspended, no data queued, task checkpointing already enabled or disabled, AST recognition already enabled or disabled, AST entry already unspecified
D.RS10==-10. Issuing task not checkpointable
D.RS16==-16. Privilege violation
D.RS17==-17. Vector already in use (CINTS)
D.RS19==-19. Illegal vector (CINTS)
D.RS80==-80. Directive issued from AST routine, directive not issued from AST routine
D.RS81==-81. Cannot map ISR or disable-interrupt routine (CINTS)
D.RS84==-84. Alignment error
D.RS85==-85. Address window overflow
D.RS86==-86. Invalid region ID
D.RS87==-87. Invalid window ID
D.RS90==-90. Specified LUN is locked in use
D.RS92==-92. Invalid device or unit specified
D.RS93==-93. Invalid time parameter
D.RS94==-94. Partition or region not in system
D.RS95==-95. Invalid priority
D.RS96==-96. Invalid LUN

D.RS97==-97. Invalid EFN or required EFN not specified
D.RS98==-98. Part of DPB is outside of issuing task's address space
D.RS99==-99. Invalid DIC or DPB size
D.RS22==2. EFN was set
D.RS00==0. EFN was clear

Global conditional assembly definitions:
D$$YNM==0 Globally define D$$YNM if dynamic memory allocation is present
M$$EXT==0 Globally define M$$EXT if 11/70 extended memory is present
M$$MGE==0 Globally define M$$MGE if memory management is present

Macro Library Calls -
ITBDF$ Define ITB offsets and length
PCBDF$ Define partition control block offsets and length
SCBDF$ Define status control block offsets
TCBDF$ Define task control block offsets

Entry Point - None

Calls None

Input None

Output None

7.2.44 SYSTB Module

SYSTB This module defines the system tables.

Macro Library Calls -
HWDDF$ Define hardware registers
SCBDF$ Define SCB offsets
UCBDF$ Define UCB offsets

Entry Point -
$DEVTB:: This is the main entry point for the SYSTB module. It indicates the start of the device tables.

Calls None

Input None

Output None

Note SYSTB is generated by SYSGEN

7.2.45 SYSXT Module

SYSXT SYSXT performs system entrance, exit and processor dispatching. SYSXT contains the following routines:
$DIRSV:: Directive save routine
$FORK:: Fork and create system process
$FORK1:: Fork and create system process
$FORK0:: Fork and create system process

$FORK2::	Fork routine to use with the CINT$ directive
$INTXT::	Interrupt exit processing
$INTSC::	Interrupt save (interrupt connected to via a CINT$ directive)
$INTSE::	Interrupt save (error logging devices)
$INTSV::	Interrupt save
$INTXT::	Interrupt exit
$DIRXT::	Directive exit
$NS0:: - $NS7::	Nonsense interrupt entry (if error logging of undefined interrupts is not supported, all unused vectors point to the nonsense interrupt address).
$NONSI::	Nonsense interrupt exit

Fork routines are entered via a CALL with the arguments:
R3 = Address of the beginning of the fork block+2
R4 = Restored from fork block
R5 = Restored from fork block

Execute fork routine: (20$:) Removes entry form fork queue, resets fork queue listhead, and allows interrupts, restores registers R4 and R5, calls the fork routine, and branches to $DIRXT to try to exit again.

Rescheduling or powerfail routine: (40$:) Allows interrupts, tests for power failure. If power failure, executes a CALL to $POWER for power recovery and then branches to $DIRXT to try to exit again. If not power failure, goes to rescheduling routine.

RESCH:	Rescheduling routine
$FINBF::	Finish terminal input buffered I/O
$SAVNR::	Save non-volatile registers
$SWSTK::	Switch stacks

Macro Library Calls -

ABODF$	Define task abort codes
HDRDF$	Define task header offsets
HWDDF$	Define hardware registers
PCBDF$	Define partition control block offsets
TCBDF$	Define task control block offsets
ITBDF$	Define interrupt transfer block offsets

Entry Point -

$DIRSV:: Directive level trap service routines call $DIRSV. The stack depth is +1, thus a switch to the system stack is always necessary. At the end of trap processing, a RETURN is executed to exit from system code.

Calls (R5) to call synchronous trap routine

Input 4(SP) = PS word pushed by trap
2(SP) = PC word pushed by trap
0(SP) = Saved R5 pushed by "JSR R5,$DIRSV"

Output $DIRSV pushes R4 onto the current stack and executes a switch to the system stack. $DIRSV pushes R3 through R0 on the system stack, sets the new processor priority and calls the calling routine.

Entry Point -
$FORK:: An I/O driver calls this routine to create a system process that returns to the driver at stack depth zero (0) to finish processing. $FORK saves R4 in the controller fork block. It points to the controller block, disables timeout, and points to the end of the fork block.

Calls None

Input R5 = Address of the UCB for the unit being processed

Output None

Entry Point -
$FORK1:: This routine is an alternate entry to $FORK to create a system process and save R5. This routine consists of one instruction "MOV R5,-(R4)" that follows in line with the $FORK routine code.

Calls None

Input R4 = Address of the last word of a 3-word fork block plus 2
R5 = Data to be saved in the fork block

Output None

Entry Point -
$FORK0:: The $FORK0 entry point is a continuation of $FORK1 code. $FORK0 sets the fork PC, saves current processor priority, locks out interrupts, and links the system process to the fork queue. $FORK0 then restores processor priority and executes a RETURN if C$$INT is defined. If C$$INT is not defined, the return is not executed and the code falls into the $FORK2 routine.

Input R4 = Address of the last word of a 2-word fork block plus 2

Output None

Entry Point -
$FORK2:: $FORK2 is the fork routine for use with the CINT$ directive. $FORK2 tests to see if the fork block is already in use (fork PC non-zero). If it is, $FORK2 clears the stack and falls through to $INTXT, which executes a return. Otherwise, $FORK2 saves R4 in the fork block, points to the location just after the three word fork block and branches to $FORK1. The combination of $FORK1, $FORK0, and $FORK2 create a system process, link the new fork entry in the fork queue, restore processor priority, and end up at $INTXT for the RETURN.

Calls None

Input R5 = Address of fork block in ITB

Output If the fork block is not in use, $FORK2 saves R4 in the fork block, puts a pointer in R4 to point just after the 3-word fork block, and clears the stack.

Entry Point -

$INTXT:: This is the interrupt exit routine. It contains only a RETURN instruction. A JUMP to this entry point causes an exit from an interrupt.

Calls None

Input 0(SP) = Interrupt save return address

Output None

Entry Point -

$INTSC:: Interrupt save routine (for interrupt from vector connected to via a CINT$ directive). $INTSC saves R4 on the stack and checks for $STKDP=0. If $STKDP=0, $INTSC loads ISR priority, calls ISR, and branches to $INTX1 to exit from interrupt. If $STKDP not = 0, $INTSC saves stack pointer in header, loads the system stack pointer, loads ISR priority, calls ISR, and branches to $INTX1 for interrupt exit.

Calls @(R5)+ to call ISR

Input None

Output None

Entry Point -

$INTSE:: This is the interrupt save routine for error logging devices. An interrupt service routine calls $INTSE when an interrupt is not to be immediately serviced. $INTSE saves R4 and thens loads R4 with the address of the SCB of the controller that caused the interrupt. $INTSE then checks if an error is already in progress. If not, $INTSE saves the current I/O active bitmap and loads R4 with the controller index. $INTSE code finishes in the $INTSV code to execute an interrupt save.

Calls None

Input
```
4(SP) = PS word pushed by interrupt
2(SP) = PC word pushed by interrupt
0(SP) = Saved R5 pushed by "JSR R5,$INTSE"
  R5  = Address of the SCB of interrupting controller
2(R5) = New processor priority
```

Output R4 = Controller index
The bit is cleared in the bitmap.

Entry Point -

$INTSV:: This is the interrupt save routine. An interrupt service routine calls this routine when an interrupt is not to be immediately dismissed. $INTSV switches to the system stack if the current stack depth is +1. When the interrupt service routine is finished processing, it forks, jumps to $INTXT, or executes a return.

Calls 2(R5) to call the caller back
(R5) to call the caller back if L$$SI1 is not defined

Input
```
4(SP) = PS word pushed by interrupt
2(SP) = PC word pushed by interrupt
```

0(SP) = Saved R5 pushed by "JSR R5,$INTSV"
R5 = New processor priority

Output

Switch to system stack if stack depth is +1

New processor priority is loaded

Entry Point -
$INTX1::

This is the interrupt exit routine. This routine is entered from a return to exit from an interrupt. $INTX1 locks out interrupts and if the stack depth is not = 0, $INTX1 branches to $DIRXT to increment the stack depth and restore registers R4 and R5; then, $DIRXT executes an RTI instruction. If the stack depth is zero, $INTX1 checks for entries in the fork queue. If the fork queue is empty, $INTX1 branches to $DIRXT to increment the stack depth, restore registers R4 and R5, and execute an RTI instruction. If the fork queue is not empty, $INTX1 allows interrupts, saves registers R0-R3 on the current stack, and proceeds to $DIRXT directive exit code.

Calls

None

Input

06(SP) = PS word pushed by interrupt
04(sp) = pc word pushed by interrupt
02(SP) = Saved R5
00(SP) = Saved R4

Output

None

Entry Point -
$DIRXT::

This is the directive exit processing routine. A directive processing routine or a trap service routine use a JUMP to enter this routine for exit. If there are any entries in the fork queue, $DIRXT removes the first entry and executes the fork routine. If there are no entries in the fork queue, $DIRXT checks to see if redispatching of the processor is necessary. If not, $DIRXT restores R0-R5 and executes an RTI instrution. If processor redispatching is necessary, the processor is redispatched and $DIRXT executes the exit sequence again.

Calls

@-(R3) to call fork routine
$POWER to call power recovery routine
$ABCTK to abort current task
$QRMVF to remove ast entry from queue
$ACHCK to address check stack space
$RELOM to relocate and map stack address
@(R0)+ or @-2(R0) to call dequeue subroutine
$DEACB to deallocate control block
$DREXT to force task exit

Input

Inputs for mapped system:
16(SP) = PS word pushed by interrupt or trap
14(SP) = PC word pushed by interrupt or trap
12(SP) = Saved R5
10(SP) = Saved R4
06(SP) = Saved R3
04(SP) = Saved R2
02(SP) = Saved R1
00(SP) = Saved R0

Inputs for unmapped system:
06(SP) = Saved R3
04(SP) = Saved R2
02(SP) = Saved R1
00(SP) = Saved R0

Output — For outputs of the $DIRXT routine, consult the interrupt processing logic diagrams in this manual.

Entry Point -
$FINBF:: — This routine finishes terminal input buffered I/O. It is called to finish a buffered terminal input request that has been placed in the AST queue.

Calls — $RELOC to relocate I/O status block
$IOFIN to finish I/O operation
$DEACB to deallocate input buffer

Input — R0 = Address of I/O packet

Output — $FINBF transfers the buffered I/O to the user task and calls $IOFIN to finish the I/O request.

Entry Point -
$SAVNR:: — This is a co-routine that saves registers R4 and R5.

Calls — @(SP)+ to call the caller

Input — R4 and R5, which are the registers to be saved.

Output — $SAVNR saves R4 and R5 on the stack.

Entry Point -
$SWSTK:: — A task calls this routine to switch to the system stack, thus inhibiting task switching. The calling task must be privileged if running in a mapped system and mapped to the exec. Control is passed here from $DRDSP after the trap has occured and $DIRSV has been called.

The calling sequence is:

```
EMT      376       Trap to $EMSST in DRDSP
.WORD    ADDR      Address for return to user state
```

Calls — @(SP)+ to call the calling routine

Input — R3 = Address of PC word of trap on stack +2

Inputs for a mapped system:
22(SP) = PS pushed by trap
20(SP) = PC pushed by trap
16(SP) = Saved R5
14(SP) = Saved R4
12(SP) = Saved R3
10(SP) = Saved R2
06(SP) = Saved R1
04(SP) = Saved R0
02(SP) = Return address for system exit
00(SP) = 104376

Inputs for an unmapped system:
10(SP) = Saved R3
06(SP) = Saved R2
04(SP) = Saved R1

02(SP) = Saved R0
00(SP) = Return address for system exit

Output $SWSTK calls the user back on the system stack with all registers preserved. To return to task level the user executes a return.

7.2.46 TDSCH Module

TDSCH This module processes time dependent scheduling and device time-outs.

Macro Library Calls -

CLKDF$	Define clock queue control block offsets
HDRDF$	Define task header offsets
HWDDF$	Define hardware registers
TCBDF$	Define task control block offsets
PCBDF$	Define partition control block offsets

Entry Point -
$CKINT:: A clock interrupt causes the system to enter this routine. $CKINT calls $INTSV to save R4 and R5 and increments the interrupt count. If the result is non-zero, $CKINT executes a JUMP to $INTXT. If the count is zero, $CKINT calls $FORK0 to execute fork and process clock interrupts.

Calls $INTSV to save registers and set priority
$FORK0 to execute fork and process clock interrupts

UPTIM: - Update absolute and real time of day and date:
None

TDS: - Time dependent scheduling:
$EXRQN to clear stop bit and reallocate partition

Single-shot internal system subroutine (type 6 or 8):
@C.SUB(R4) to call system subroutine

Mark time request:
$SETM to set event flag
$QASTT to queue AST to task

Schedule request:
$TSKRT to request task execution
$DECLK to deallocate control block
$CLINS to reinsert entry in clock queue
@(SP)+ to get next UCB address

@D.VOUT(R1): - Call driver at timeout entry point with the arguments:
R0 = Device timeout status "IE.DNR"
R2 = Address of device CSR
R3 = Controller index
R4 = Address of the status control block
R5 = Address of the unit control block

ROBIN: - Executive level round robin scheduling
$DRDSE cause a redispatch of processor

SWAP: - Disk swapping algorithm; reduce sapping priority of resident tasks

$NXTSK Reallocate partition

TIMXT: - Exit time dependent scheduling if no unprocessed clock ticks remain

Input None

Output none

CHAPTER 8

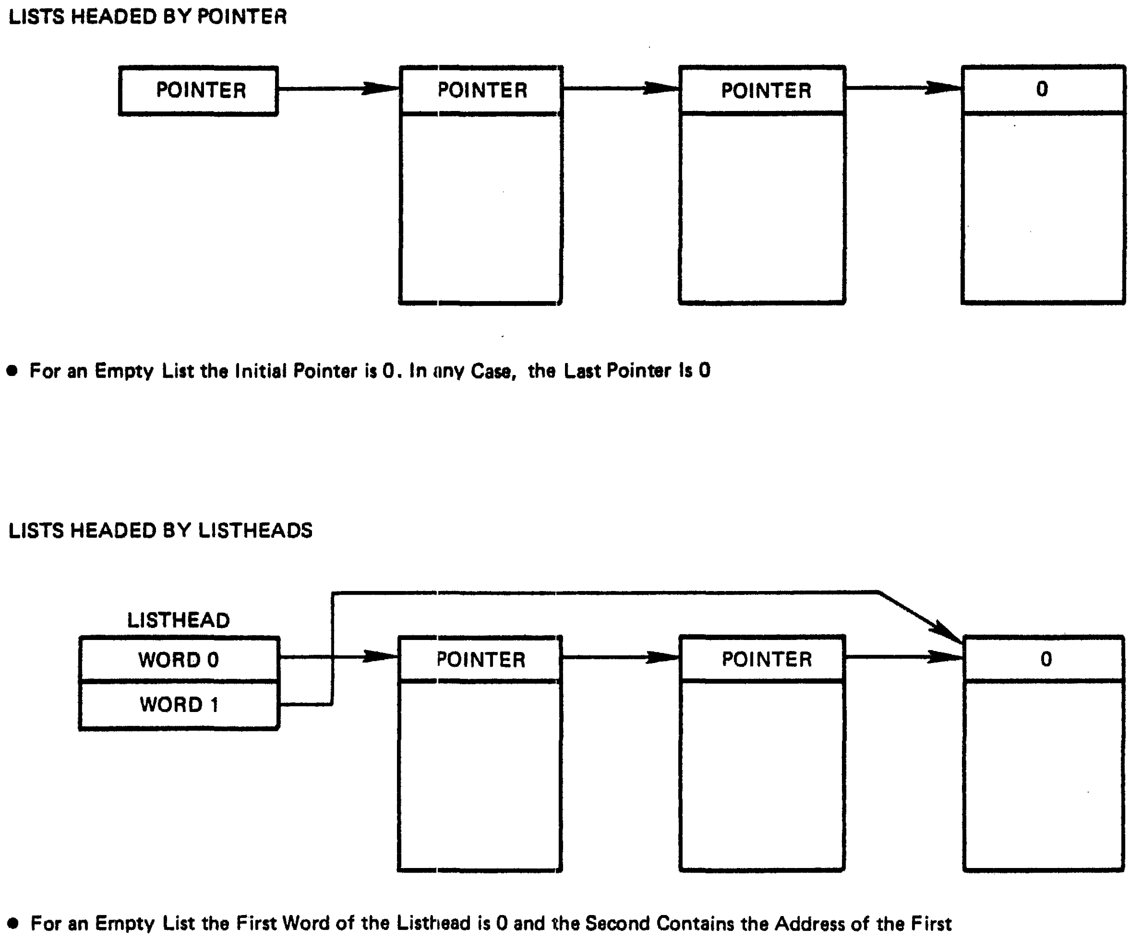
DATA AREAS AND CONTROL BLOCKS

8.1 INTRODUCTION

This chapter describes the system control block linkages and data structures. The control blocks were taken from system code. However, while transposing the system code into this manual, editorial changes were made for the sake of appearance and clarity. The beginning of the chapter describes system pointers and typical system linkages followed by a discussion of I/O linkages as related to I/O drivers. The remainder of the chapter contains the system control block offset descriptions and describes the system control blocks and data areas in alphabetical order.

8.2 SYSTEM POINTERS AND LINKAGES

Figure 8-1, Linked Lists on RSX-11M, describes the typical way that control block lists are linked in RSX-11M. Figure 8-2, Overview of RSX-11M System Control Blocks, contains an overview of the way that RSX-11M system control blocks are linked together. This figure shows the major system lists, list pointers, and linkages only. More detailed linkages and control block configurations are shown in Figures 8-3 through 8-24.



LISTS HEADED BY POINTER

- For an Empty List the Initial Pointer is 0. In any Case, the Last Pointer Is 0

LISTS HEADED BY LISTHEADS

- For an Empty List the First Word of the Listhead is 0 and the Second Contains the Address of the First

Figure 8-1 Linked Lists on RSX-11M

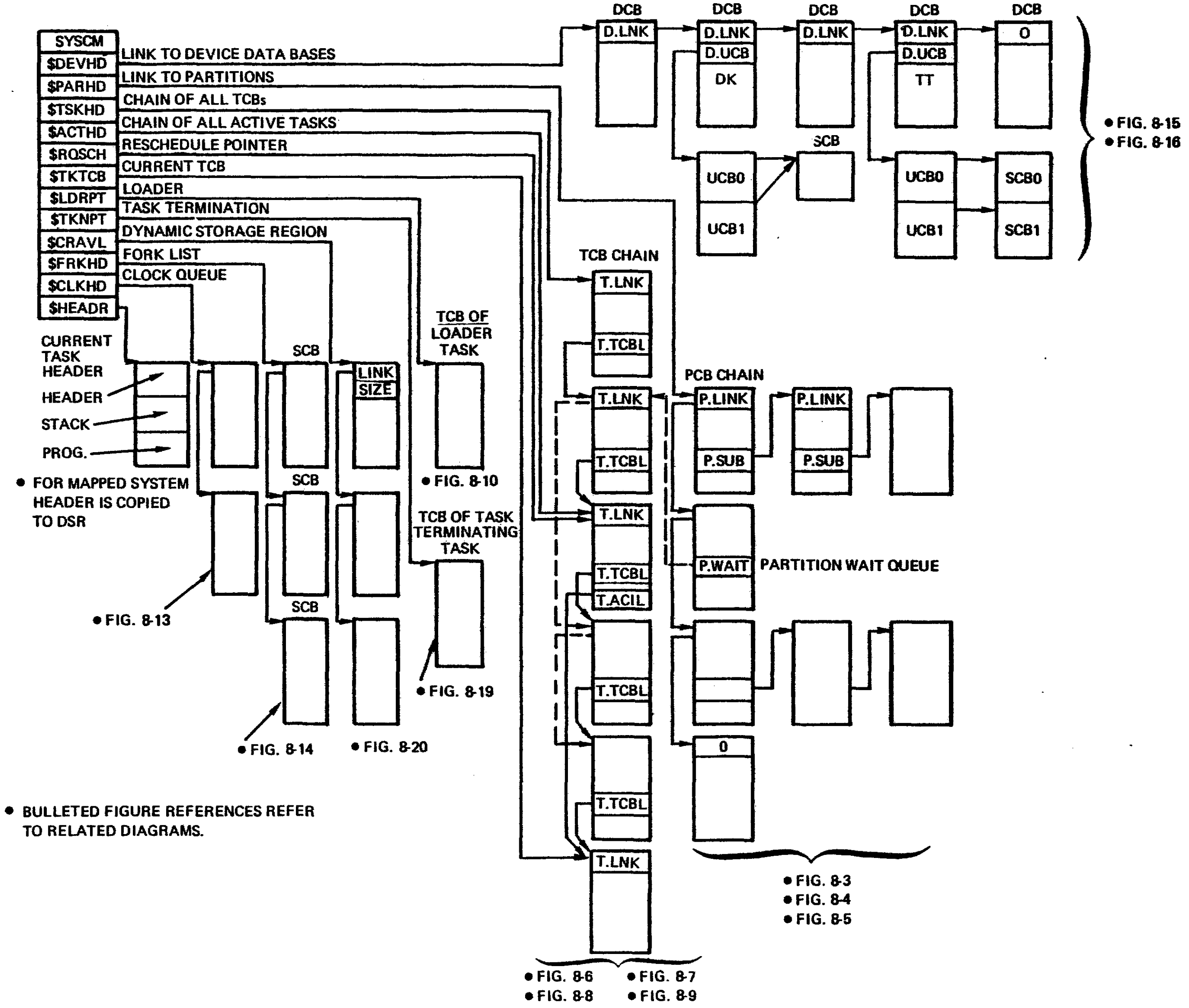


Figure 8-2 Overview of RSX-11M System Control Blocks

8.2.1 Device Control Block Pointer ($DEVHD)

The location, $DEVHD, which is in SYSCM, contains the address of the label, $DEVTB. $DEVTB is the label of the first DCB of a list of DCBs in the system.

SYSGEN produces SYSTB, which contains the DCBs, UCBs, and SCBs for the system as well as the label, $DEVTB::. The Task Builder resolves the references to the first DCB ($DEVTB) when it links SYSTB into the Executive.

The DCBs are linked by the first word in each DCB, D.LNK. D.LNK contains the address of the next DCB in the chain. Typically, when the Executive works with the DCB, R3 contains the address of the DCB. To access the next DCB, the Executive executes a MOV @R3,R3 instruction.

The DCBs and UCBs in SYSTB contain the logical name, logical unit number, and device characteristics of the devices in the system. In general, the DCB contains device-unit information common to all device-units of the same type on a controller. This kind of structure saves space because, otherwise, this common information would have to appear in the UCB for each device-unit.

The DCB contains the device name; for example, the DCB for DK0: contains "DK" in ASCII. With one exception, there is only one DCB with the name of a given device in it. The exception occurs when there is a DCB for every kind of terminal device controller in the system (DL, DJ, DH, or DZ); however, they all contain the device name "TT".

There are two types of DCB; one for real devices and one for pseudo devices. Nothing in the DCB denotes its use for a real or pseudo device. However, the UCB associated with the DCB contains the bit, DV.PSE, which identifies the associated device as a pseudo device.

An Executive co-routine, $SCNDT, is used to scan all the addresses of the DCBs and UCBs in the system up to but not including the first pseudo device. For example, the address of the DCB for TI: cannot be found by using this co-routine.

8.2.2 Unit Control Blocks

A UCB exists for each device-unit on the system. D.UCB, which is a word in the DCB, contains a pointer to the first UCB for that DCB. There is at least one UCB for each DCB; however, there can be more than one UCB for each DCB.

For an example of a DCB-UCB relationship, a system has four identical disk drives on one controller. The disk device-units would have physical unit numbers from 0 through 3 and logical unit numbers from 0 through 3. In this configuration, there is one DCB (all device-units are the same), four UCBs (one for each device-unit), and one Status Control Block (SCB) for the controller. In this example, only one drive can be operated at a time.

For another example, a system has four identical disk drives; two connected to controller 0 and two connected to controller 1. The two drives on controller 0 have physical unit numbers 0 and 1, and logical unit numbers 0 and 1. The two drives on controller 1 have physical unit numbers 0 and 1, and logical unit numbers 3 and 4. In this configuration, there is one DCB (all device units are the same), four UCBs (one for each device-unit), and two SCBs (one for each controller). In this example, two drives can operate in parallel as long as they are on different controllers: that is, the drives with logical unit numbers 0 and 2 or 0 and 3, or the drives with logical unit numbers 1 and 2 or 1 and 3.

The UCB contains pointers and device status information.

8.2.3 Status Control Block (SCB)

There is one Status Control Block for each controller on the system and it contains controller status information. The UCB points to the related SCB. See the discussion that precedes the word and bit definitions for the Status Control Block.

8.2.4 Partition Control Block (PCB) Pointer

$PARHD is the pointer to the first PCB in the chain of system PCBs.

System- and user-controlled partitions can be differentiated by examining the form of their PCB linkage.

In a user-controlled partition PCB, P.LNK of this PCB points to the first word, P.LNK, of the next subpartition PCB. P.SUB in the first PCB also points to P.LNK of the next PCB. P.MAIN, which is in user-controlled partition and subpartition PCBs, points back to the main partition PCB that heads the list of subpartition PCBs. In a user-controlled main partition PCB without subpartitions, P.SUB is 0. However, if the main partition has subpartitions, P.SUB in the last subpartition PCB is zero.

This linkage is slightly different in system-controlled partitions. The dynamically created subpartition PCBs are not linked by P.LNK; only P.SUB points to the first word, P.LNK, of the next PCB in the chain of PCBs. P.MAIN in each system-controlled partition or subpartition points back to the main PCB. In a PCB for a system-controlled partition without dynamically created subpartitions. P.LNK is 0. If the partition has subpartitions, P.SUB in the last subpartition PCB is 0.

See Figure 8-3 for an example of a system- and user-controlled partition list.

8.2.5 Task Control Block (TCB) Pointers ($TSKHD And $ACTHD)

Task Control Blocks (TCBs) are listed in several lists. One is the System Task Directory (STD), a list (ordered by priority) of all installed tasks. Another is the Active Task List (ATL), a list of tasks that are currently active. The ATL is a subset of the STD; therefore, TCBs that are linked into the ATL are also linked into the STD.

$TSKHD points to the first TCB in the STD. The remaining TCBs in the STD are linked by the word, T.TCBL. $TSKHD points to T.LNK in the first TCB, T.TCBL in the first TCB points to T.LNK in the next TCB, and so on.

$ACTHD points to the first TCB in the ATL. The word, T.ACTL, links the remaining active task TCBs. T.ACTL in the last TCB word is 0; this indicates the TCB of the Null task.

The Partition Wait Queue is a list of TCBs of those tasks waiting to use a given partition. The word, P.WAIT, in the PCB points to the first TCB in the Partition Wait Queue. T.LNK in the first TCB points to the T.LNK word in the second TCB, T.LNK in the second TCB points to the T.LNK word in the third TCB, and so on.

8.2.6 Reschedule Pointer ($RQSCH)

The reschedule pointer, $RQSCH, contains the address of the TCB of the task to be rescheduled.

8.2.7 Current Task Pointer ($TKTCB)

The current task pointer, $TKTCB, contains the address of the TCB of the currently running task.

8.2.8 Loader Pointer ($LDRPT)

Normally, $LDRPT is the pointer to the TCB of the Loader task.

8.2.9 Task Termination Task Pointer ($TKNPT)

$TKNPT points to the TCB of the Task Termination Task.

8.2.10 Free Storage Block Pointer ($CRAVL)

$CRAVL points to the first free block in a list of free storage blocks in the Dynamic Storage Region. $CRAVL-2 is a word that contains a 3 and should never contain less than a 3. This number is one less than the number of bytes in the smallest possible block obtainable from the Dynamic Storage Region. This number 3 is a rounding factor. In other words, the size of the smallest allocatable memory block is 4 bytes and memory must be allocated in multiples of 4 bytes. Four bytes are needed because every free memory block must contain a pointer to the next block, followed by its own size in bytes. This information uses up 2 words of the block.

Initially, the Dynamic Storage Region is a continuous memory area. However, it becomes somewhat fragmented after the system begins execution.

8.2.11 Fork Queue List Pointer ($FRKHD)

$FRKHD is a pointer to a list of SCBs in the fork queue.

8.2.12 Clock Queue Pointer ($CLKHD)

$CLKHD is a pointer to a list of 8-word clock queue control blocks.

8.2.13 Current Task Header Pointer ($HEADR)

$HEADR points to the current task header. In a mapped system, the Executive copies the task header into the DSR. Only privileged tasks can access this copy; the Executive refers to and modifies this copy as execution proceeds. Therefore, the original header of the current task may not contain valid information because the valid header is in the DSR space. Before the task is checkpointed, the Executive replaces the original header with the copy in the DSR.

8.2.14 Examples Of System Linkages

Figure 8-3 contains an example of a PCB linkage for user- and system-controlled partitions.

Figure 8-4 shows TCB wait queues in user- and system-controlled partitions.

Figure 8-5 shows checkpoint file PCBs with and without checkpointed task TCBs.

Figure 8-6 shows an example of TCBs linked into a System Task Directory with some of the TCBs in the Active Task List.

Figure 8-7 shows a simplified linkage of the PCB, TCBs, and task header in a task partition.

Figure 8-8 shows a linkage of TCBs for resident and non-resident tasks with their respective PCBs.

Figure 8-9 shows AST control blocks in the AST queue.

Figure 8-10 shows TCBs of tasks in the Loader queue.

Figure 8-11 shows Send/Receive data blocks queued to the receiver task TCB.

Figure 8-12 shows Send/Receive-by-Reference blocks queued to the receiver task TCB.

Figure 8-13 shows the linkage of Clock Queue Control Blocks.

Figure 8-14 shows the linkage of the Fork Control Blocks.

Figure 8-15 shows an example of DCBs, SCBs, UCBs, and LCBs in system. It also shows the linkage caused by redirected and reassigned devices.

Figure 8-16 Shows the Logical Assignment Control Block linkage.

Figure 8-17 shows the linkage of MCR queue entries.

Figure 8-18 shows the linkage of pre-allocated I/O packets.

Figure 8-19 shows Message Blocks in the Task Termination Notification (TKTN) queue.

Figure 8-20 shows the linkage of the free blocks in the Dynamic Storage Region.

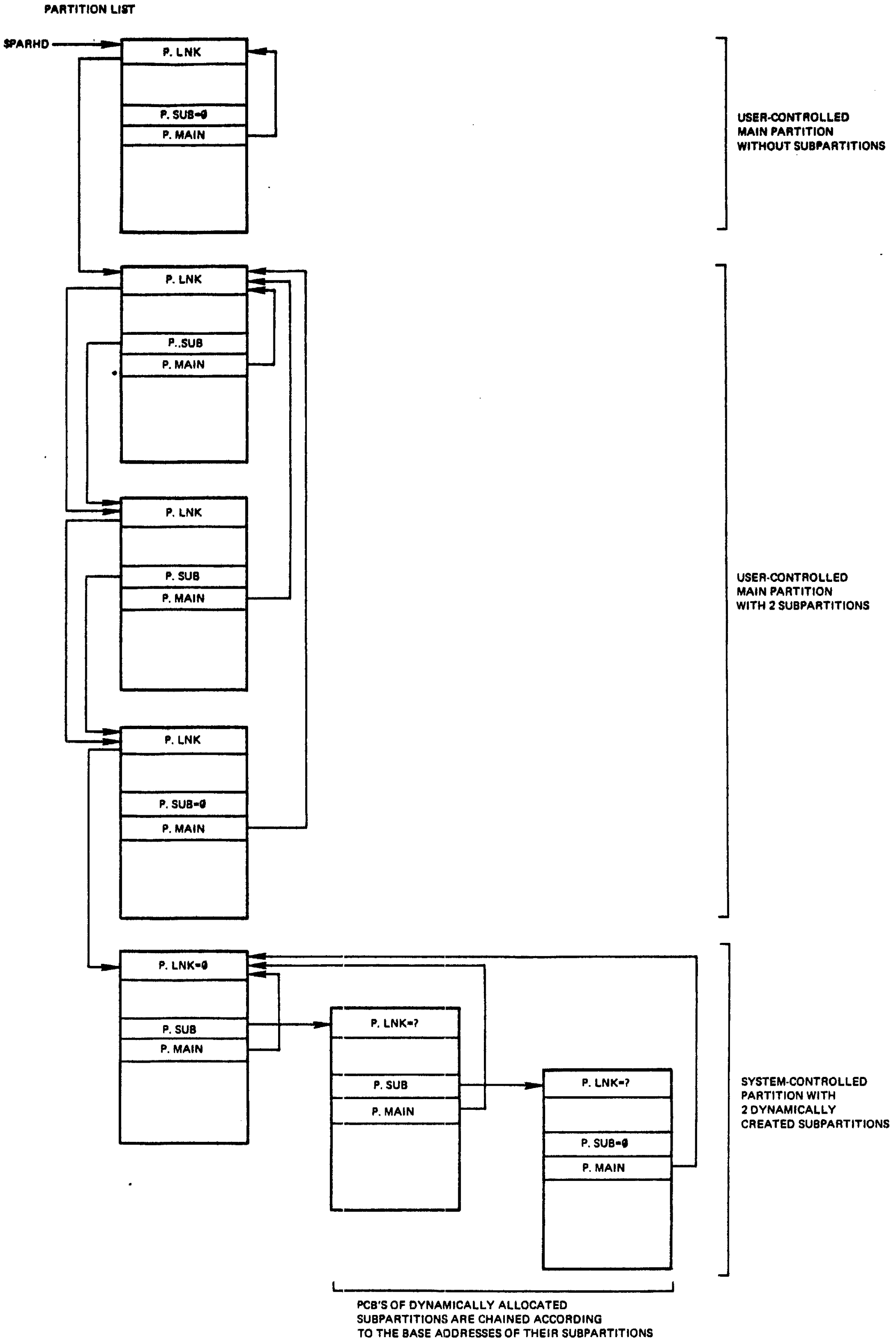
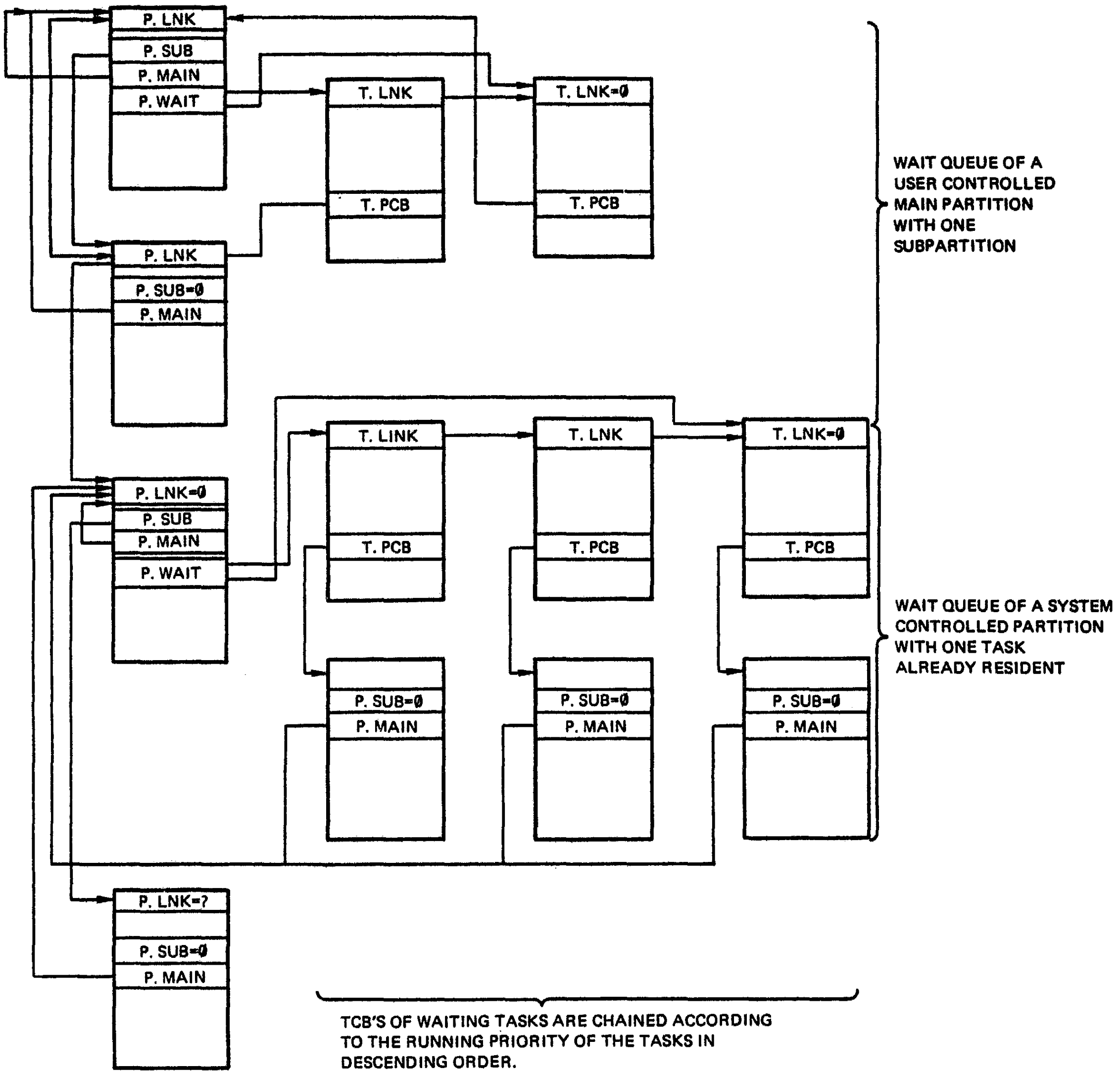

PARTITION LIST
$PARHD
P. LNK
P. SUB=0
P. MAIN
USER-CONTROLLED MAIN PARTITION WITHOUT SUBPARTITIONS
P. LNK
P..SUB
P. MAIN
P. LNK
P. SUB
P. MAIN
USER-CONTROLLED MAIN PARTITION WITH 2 SUBPARTITIONS
P. LNK
P. SUB=0
P. MAIN
P. LNK=0
P. SUB
P. MAIN
P. LNK=?
P. SUB
P. MAIN
P. LNK=?
P. SUB=0
P. MAIN
SYSTEM-CONTROLLED PARTITION WITH 2 DYNAMICALLY CREATED SUBPARTITIONS
PCB'S OF DYNAMICALLY ALLOCATED SUBPARTITIONS ARE CHAINED ACCORDING TO THE BASE ADDRESSES OF THEIR SUBPARTITIONS

Figure 8-3 Example of PCB Listings



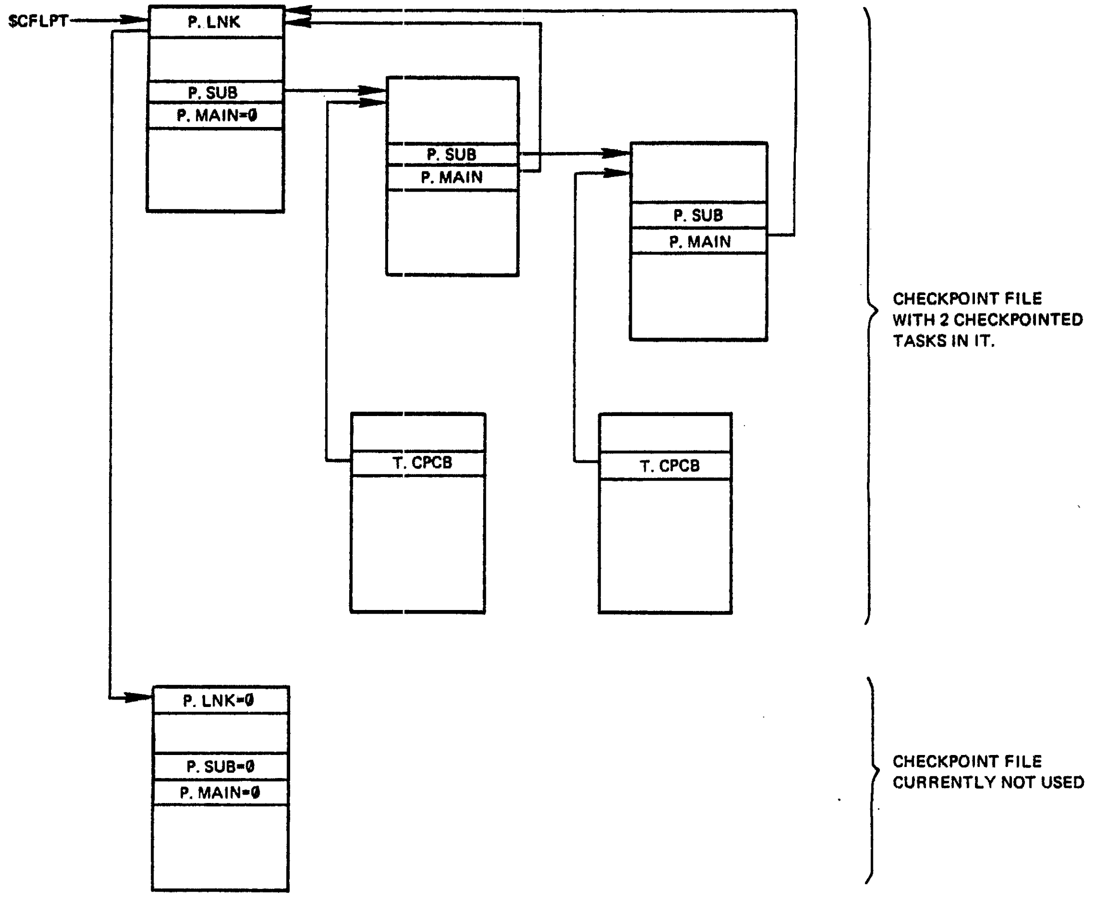
WAIT QUEUE OF A USER CONTROLLED MAIN PARTITION WITH ONE SUBPARTITION

WAIT QUEUE OF A SYSTEM CONTROLLED PARTITION WITH ONE TASK ALREADY RESIDENT

TCB'S OF WAITING TASKS ARE CHAINED ACCORDING TO THE RUNNING PRIORITY OF THE TASKS IN DESCENDING ORDER.

- The main PCB always heads the Partition Wait Queue. The Partition Wait Queue contains TCBs both of tasks to be loaded for the first time and of checkpointed tasks.

Figure 8-4 Example of a Partition Wait Queue



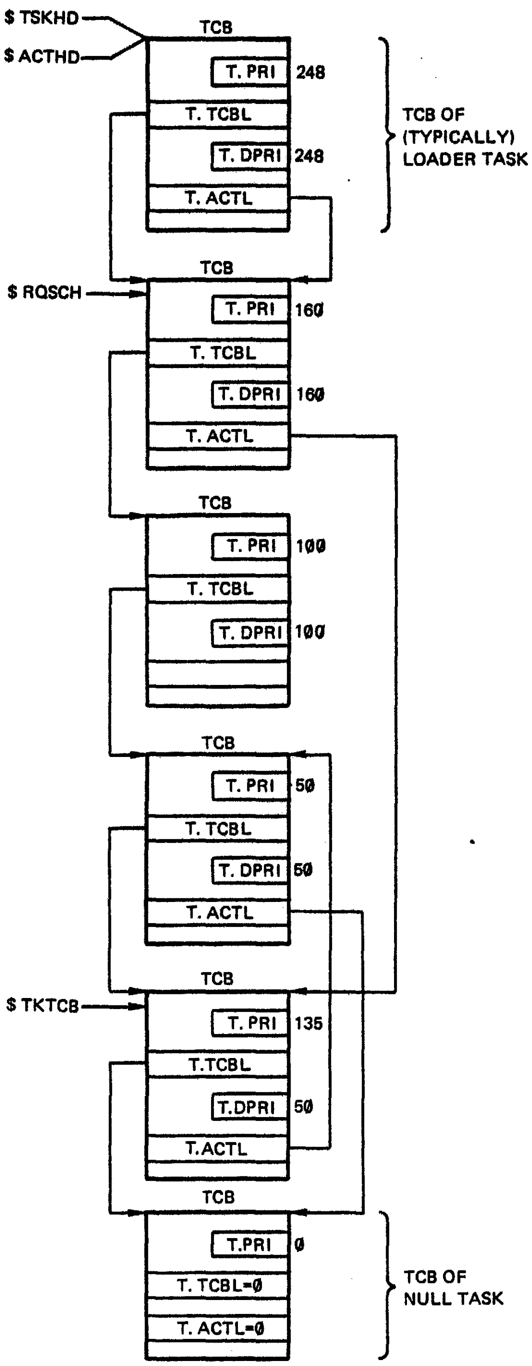
- PCBs for checkpoint files are allocated by the MCR command:

  ACS devunit: /BLKS=no. of blocks

  The PCBs are chained in the order of their allocation.

- PCBs for checkpointed tasks are dynamically allocated whenever the Executive checkpoints a task and are chained in ascending order according to base disk address.

Figure 8-5 Example of a PCB List for Checkpoint Files



- $TSKHD points to the start of the STD.
- $ACTHD points to the start of the ATL.
- $TKTCB points to the current task.
- $RQSCH points to where the Executive starts scanning the ATL.
- A task is put into the STD by the Install command and removed by the Remove command.
- The STD is linked in descending order according to the default priority.
- The ATL is linked according to the running and default priority in descending order.
- The default priority is defined when the task is installed:

  TKB-option PRI=n
  INS filespec/PRI=n
  RUN filespec/PRI=n
  default=50.
  ALT taskname/PRI=n
- The running priority is defined by:

  ALT taskname/PRI=n
  ALT taskname/RPRI=n
  ALTP$ directive
  default: default priority
- The round robin scheduler periodically reorders the tasks of the same numeric running priority within the ATL; it does not affect the STD.
- The swapping algorithm does not affect the STD or the ATL.
- The following are pointers to TCBs of special tasks:

  $LDRPT points to the Loader TCB
  $MCRPT points to the MCR TCB
  $TKNPT points to the task termination notification task TCB
  $SHFPT points to the shuffler TCB
  $ERRPT points to the error logger TCB

Figure 8-6 Example of a System Task Directory (STD) and Active Task List

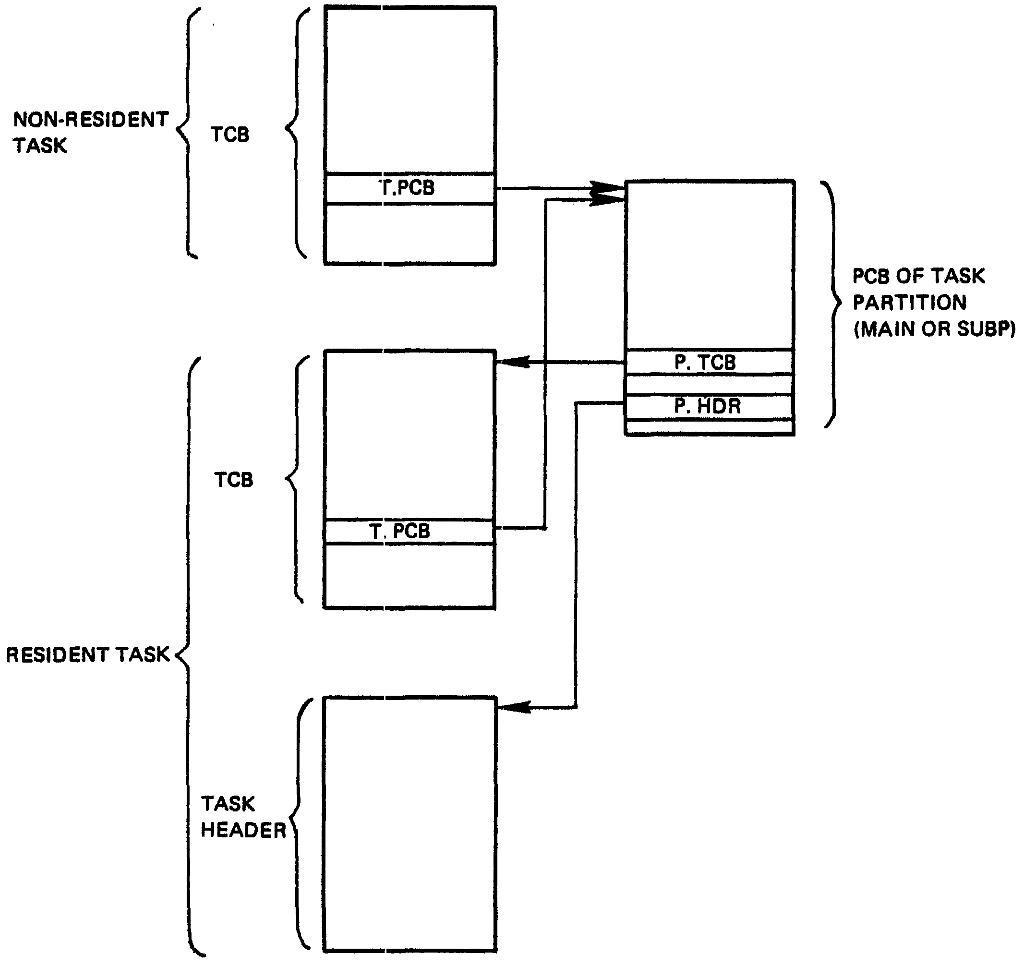


Figure 8-7 Simplified User-Controlled Partition TCB, Task Header, and PCB Relationship

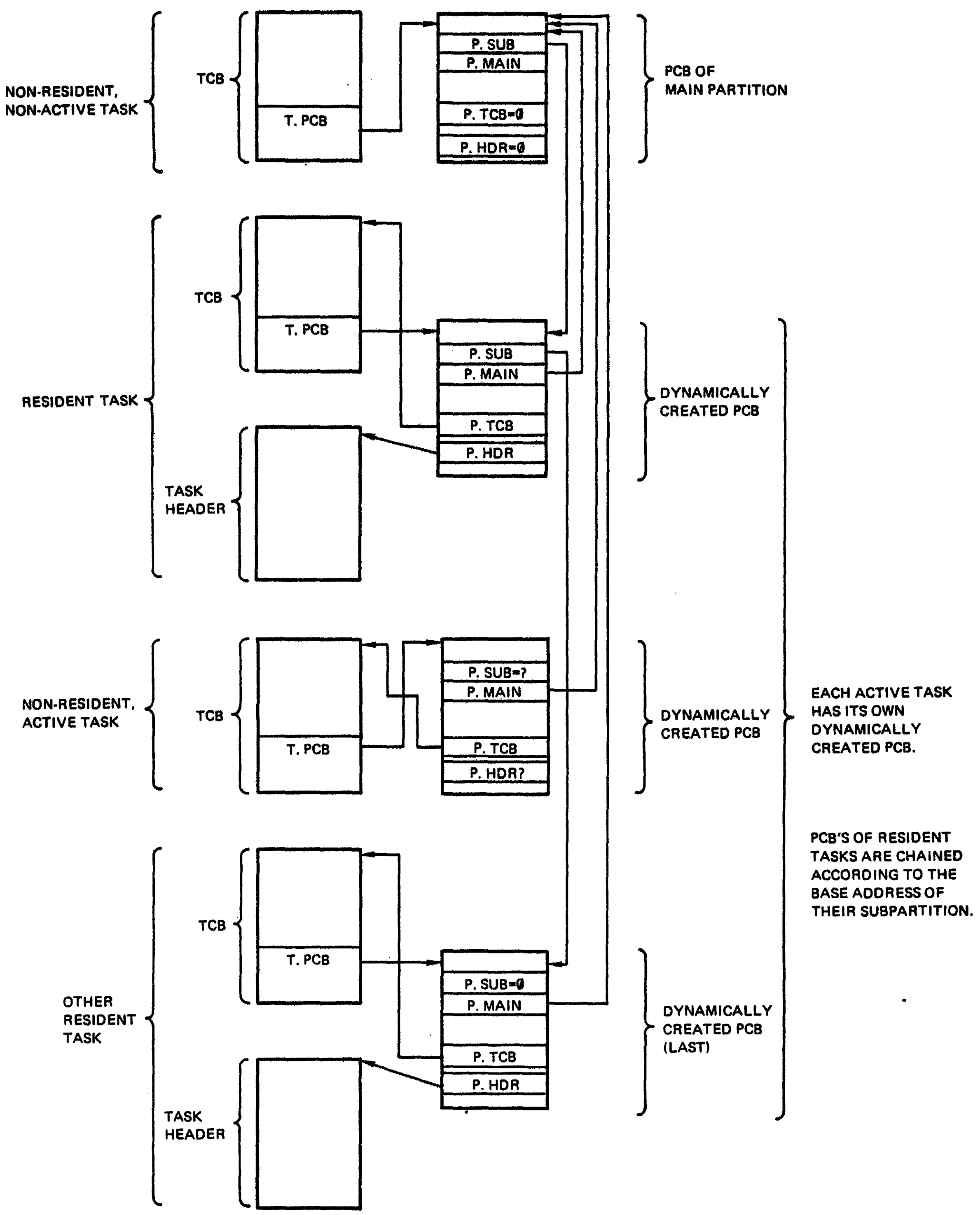
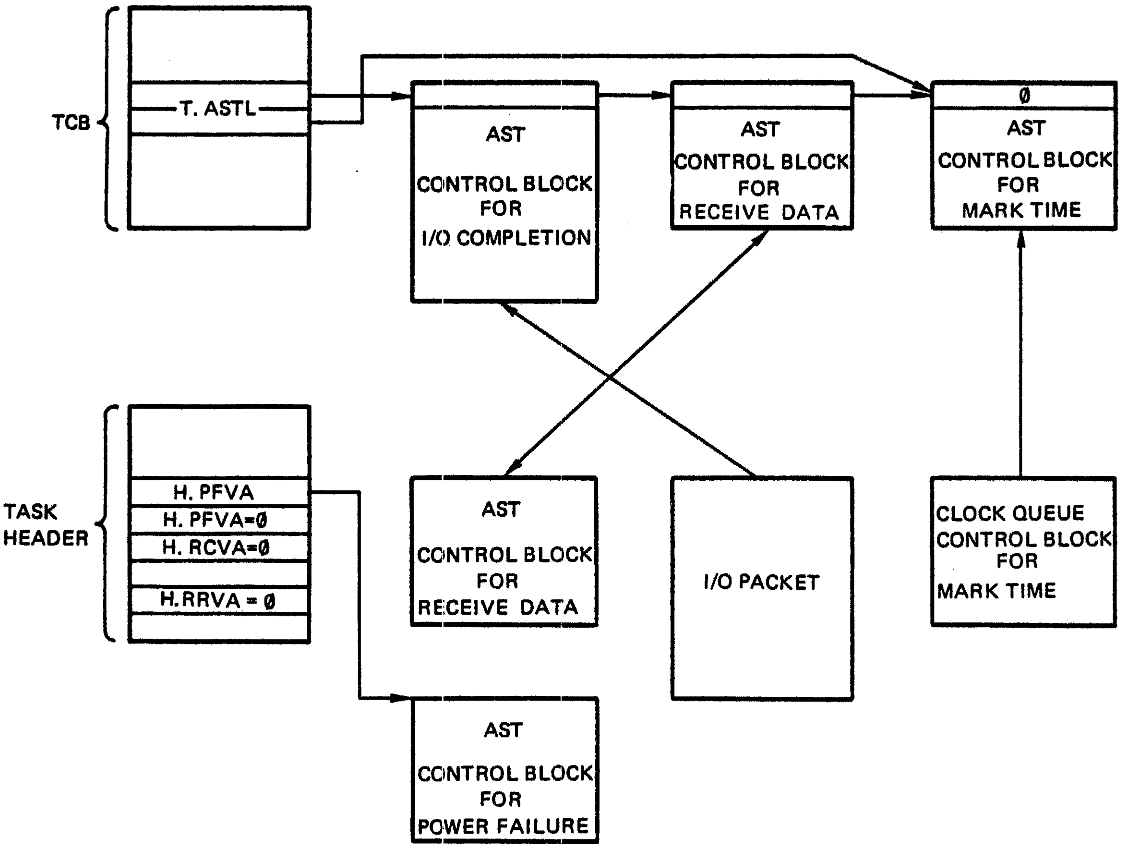


Figure 8-8 TCB, Task Header, and PCB
Relationships in a System-Controlled Partition


TCB
T. ASTL
AST CONTROL BLOCK FOR I/O COMPLETION
AST CONTROL BLOCK FOR RECEIVE DATA
Ø
AST CONTROL BLOCK FOR MARK TIME
TASK HEADER
H. PFVA
H. PFVA=Ø
H. RCVA=Ø
H.RRVA = Ø
AST CONTROL BLOCK FOR RECEIVE DATA
I/O PACKET
CLOCK QUEUE CONTROL BLOCK FOR MARK TIME
AST CONTROL BLOCK FOR POWER FAILURE

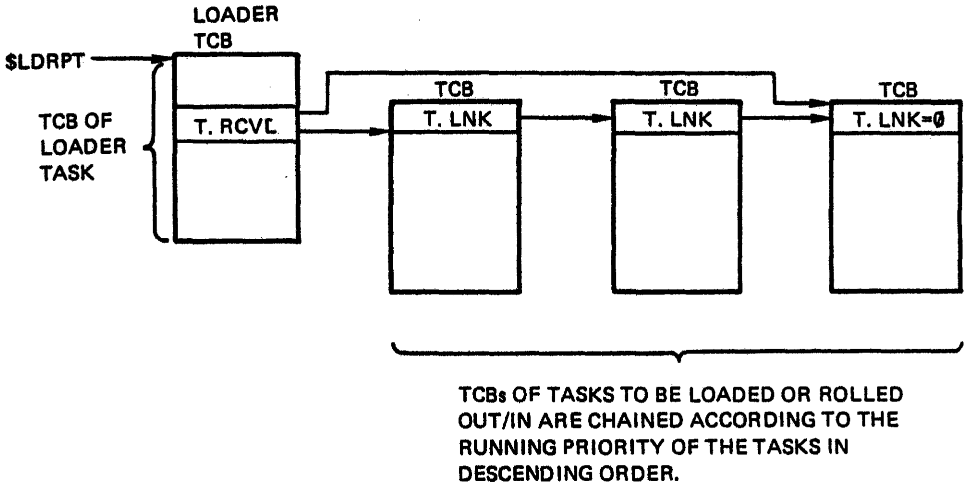
- AST control blocks are queued FIFO.

- AST control blocks for power failure, floating-point exception, Receive Data, and Receive-by-Reference are allocated a length of C.LGTH=16. bytes. They are deallocated by the STRA$, SFPA$, SRDA$, and SRRA$ directives. Their addresses are kept in the task header. When an AST is executed, the AST control block is inserted into the AST queue. After the AST has been set up, the AST control block address is again put into the task header.

- The AST control block for Mark Time is allocated a length of C.LGTH=16. bytes by the MRKT$ directive and inserted into the clock queue. When the mark time becomes due, the AST control block is removed from the clock queue and inserted into the AST queue. After the AST has been set up, the AST control block is deallocated.

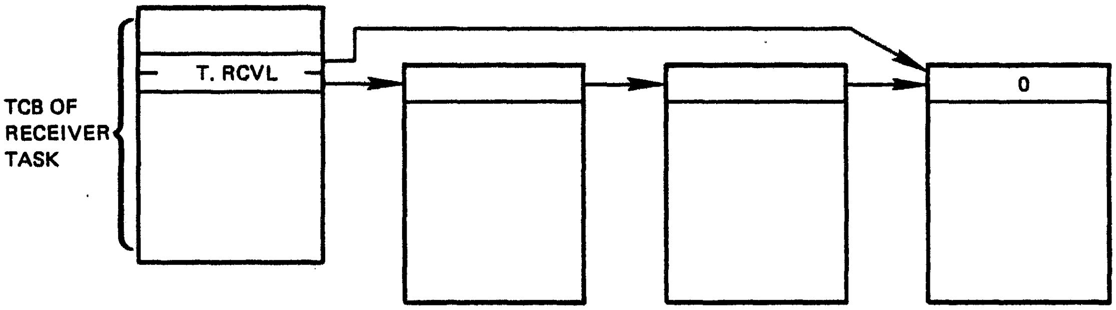
- For the I/O completion AST, the I/O packet is taken and inserted into the AST queue. After the AST has been set up, the AST control block (I/O packet) is deallocated.

Figure 8-9 Example of an AST queue


LOADER TCB
$LDRPT
TCB OF LOADER TASK
T. RCVL
TCB
T. LNK
TCB
T. LNK
TCB
T. LNK=0
TCBs OF TASKS TO BE LOADED OR ROLLED OUT/IN ARE CHAINED ACCORDING TO THE RUNNING PRIORITY OF THE TASKS IN DESCENDING ORDER.

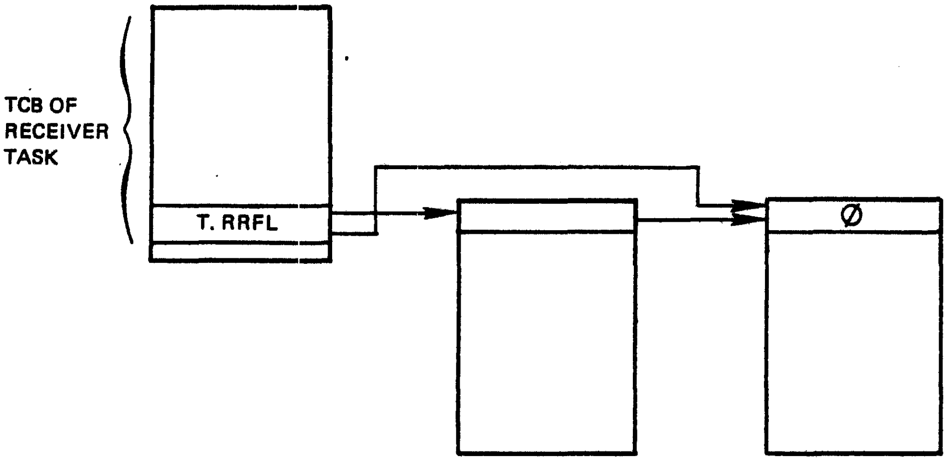
- The Loader queue is headed by the TCB of the loader task and contains TCBs of both tasks to be loaded the first time and the task to be checkpointed.

Figure 8-10 The Loader Queue



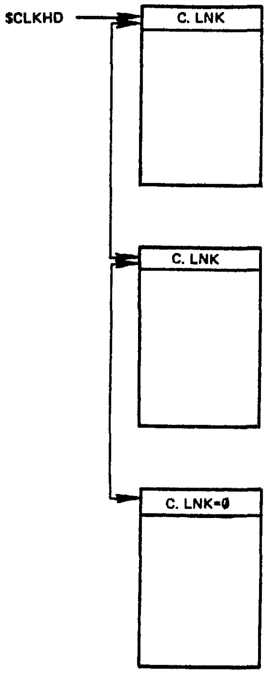
- Send/Receive data blocks are queued FIFO.
- Send/Receive data blocks are queued by issuing the SDAT$ directive and dequeued when the RCVD$ or RCVX$ directives are issued.

Figure 8-11 Send/Receive Data Queue



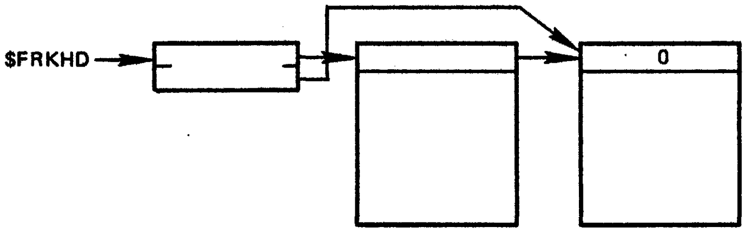
- Send/Receive-by-Reference blocks are queued FIFO.
- Send/Receive-by-Reference blocks are queued by issuing the SREF$ directive and dequeued when the RREF$ directive is issued.

Figure 8-12 Send/Receive by Reference Queue



- The clock queue is linked in ascending order according to the absolute time when the events described by the Clock Queue Control Block come due.
- The 1-word counter, $ABTIM, is incremented with each clock tick. Whenever it overflows, the high-order word of the absolute time in each clock queue is decremented.
- An event comes due when the high-order of the absolute time in its Clock Queue Control Block is 0 and the low-order word of the absolute time is less than or equal to the counter, $ABTIM.
- The Clock Queue Control Block is conditionally deallocated when the event comes due dependent on the type of Clock Queue Control Block:
  Type 0 - Used as AST control block (if
  specified), otherwise deallocated
  Type 2 - Queued again
  Type 4 - Deallocated
  Type 6, 10, or 12 - Not deallocated

Figure 8-13 The Clock Queue



- Fork control blocks are queued FIFO
- Fork control blocks are queued by issuing a $FORK, $FORK1 or $FORKØ call; the control goes back to the next higher subroutine level
- Fork processes (instructions following the CALL $FORKn instruction up to the next RETURN instruction) are executed before the system goes back to user level the next time ($STKDP is changed from Ø to 1)

FORK CONTROL BLOCK

Ø	POINTER TO NEXT FORK CONTROL BLOCK	
2	START ADDRESS OF FORK PROCESS	
4	SAVED R5	
6	SAVED R4	
1Ø	RELOCATION BIAS OF FORK PROCESS	IN MAPPED SYSTEM WITH LOADABLE DRIVER SUPPORT ONLY
12		

Figure 8-14 The Fork Queue

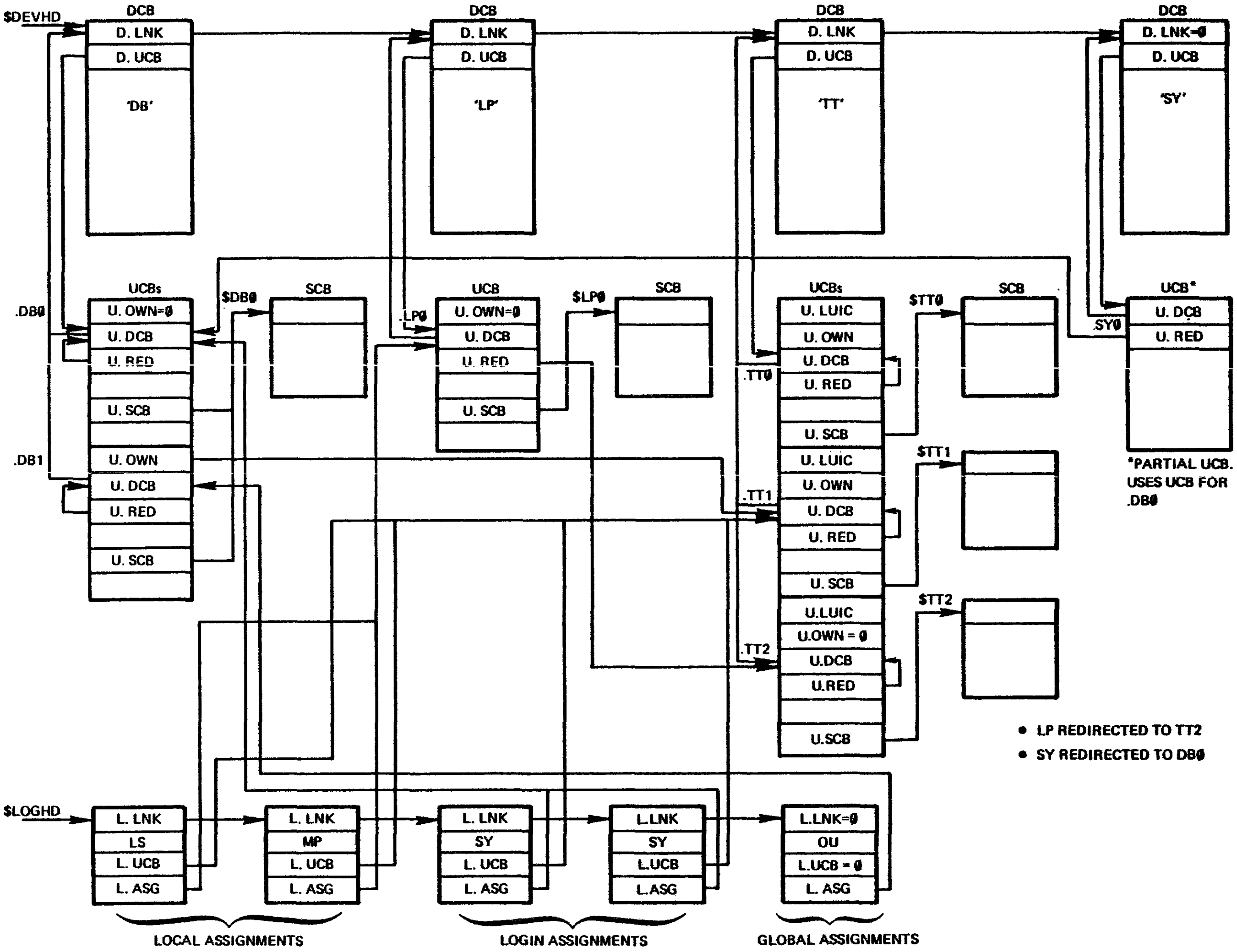
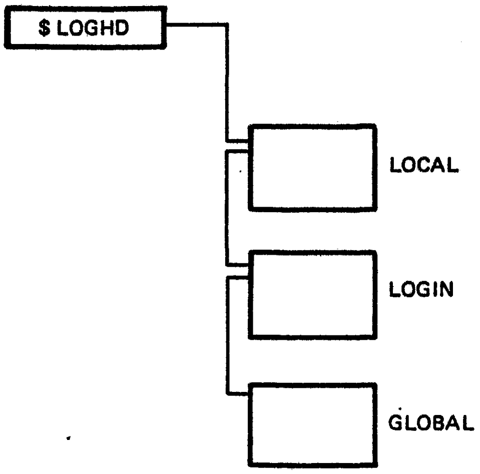


Figure 8-15 Example of DCB, SCB, UCB, LCB Relationship



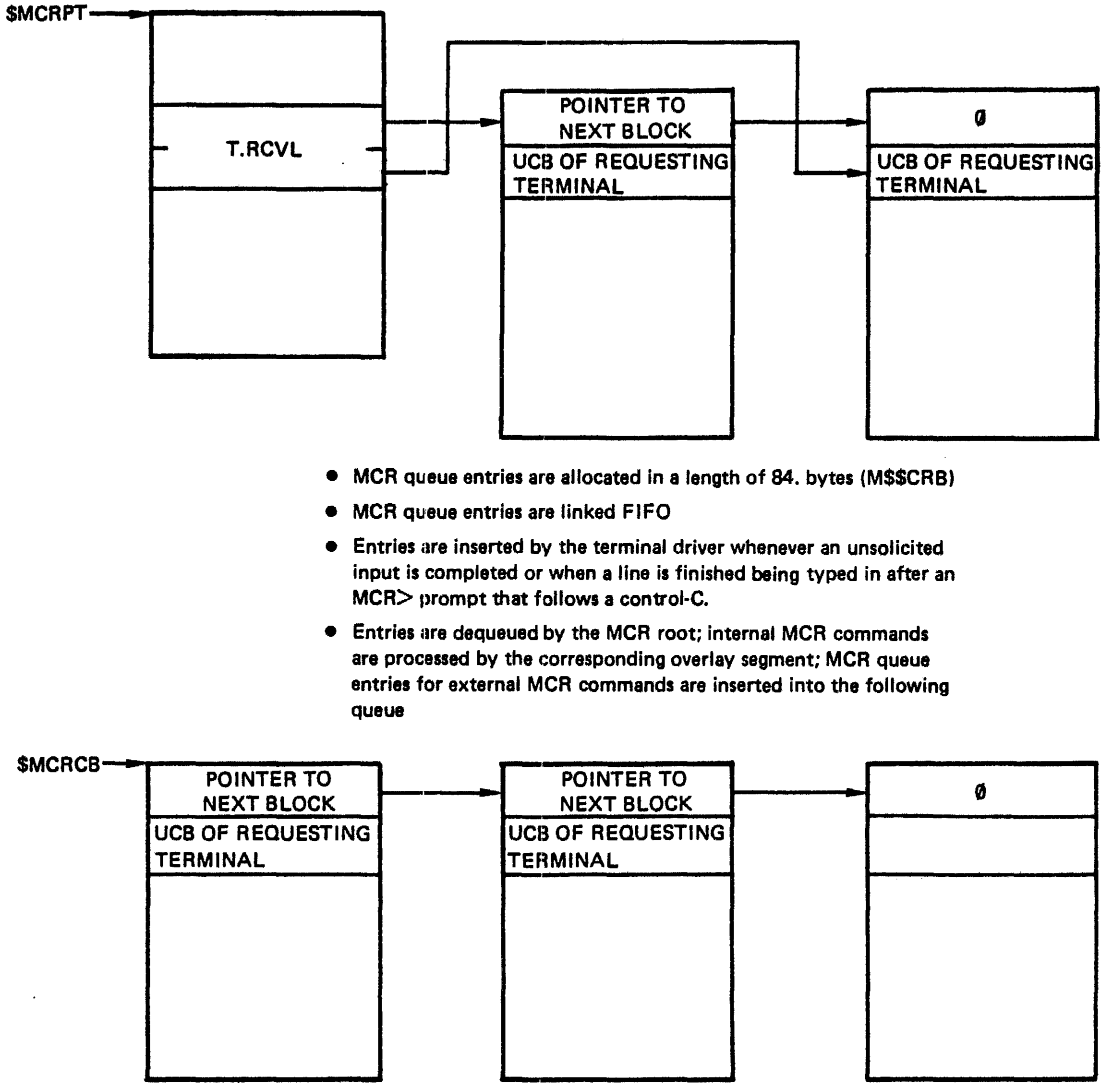
LOGICAL ASSIGNMENT CONTROL BLOCK (LCB)

	LINK TO NEXT LCB	L. LNK
	LOGICAL NAME OF DEVICE	L. NAM
L. TYPE	TYPE OF ENTRY (Ø=GLBL) LOGICAL UNIT NUMBER	L. UNIT
	TI UCB ADDRESS	L. UCB
	ASSIGNMENT UCB ADDRESS	L. ASG
		L. LGTH

- An LCB is allocated by issuing an
  ASN target device = logical device { /LOGIN [/TERM = terminal] | /GBL } command

  and deallocated by issuing an
  ASN = [logical device] { /LOGIN [/TERM = terminal] | /GBL } command
- There are three groups of logical assignments:
  1. local assignments applying to one specific terminal only
  2. login assignments applying to one specific terminal only where a user is logged in, they are established either at login time or when a login logical assignment command is issued
  3. global assignments valid for all terminals in the system
- LCBs are linked local assignments first, then login assignments and global assignments at the end; within each group they are linked according to the time the logical assignment was established
- Logical assignments are resolved by scanning the linked list of LCB's when a logical unit number is assigned to a physical device (at install time or at run time when a ALUN$ directive is issued)
- Symbolic offset definitions in LCBDF$ in [1,1] EXEMC.MLB

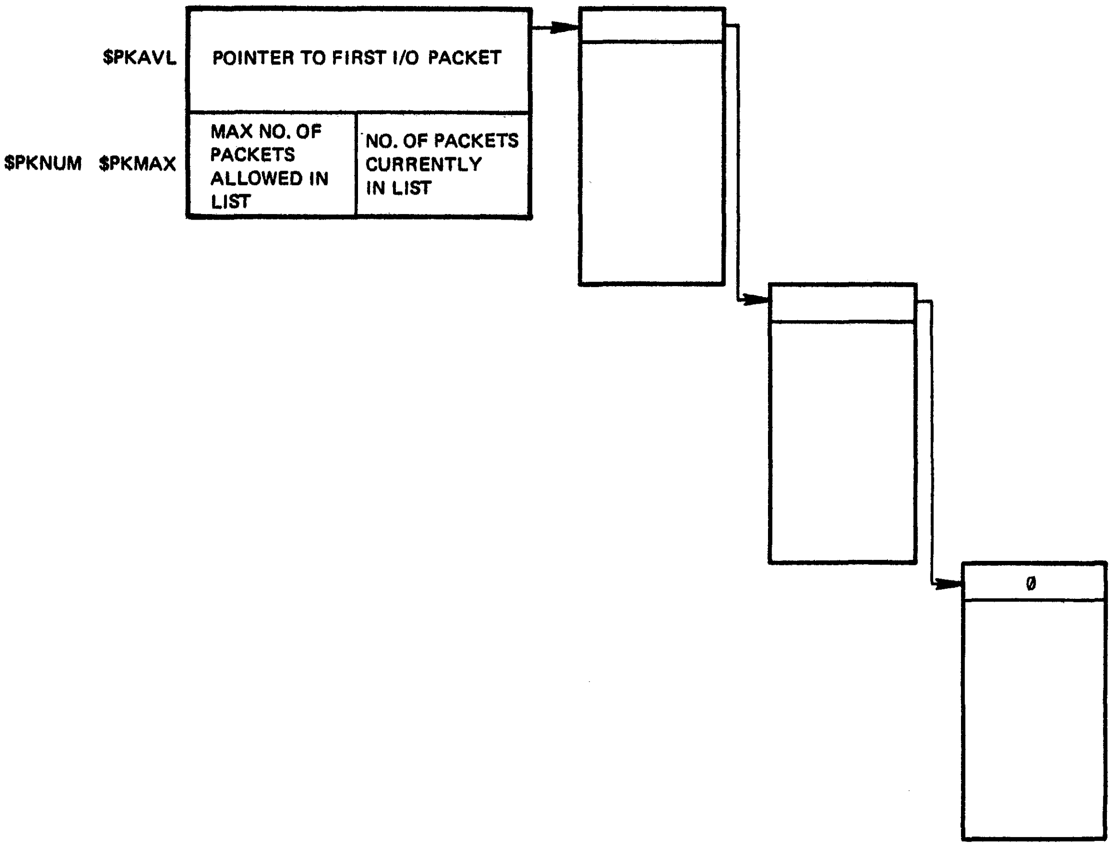
Figure 8-16 Logical Assignment Control Block (LCB) List


$MCRPT
T.RCVL
POINTER TO NEXT BLOCK
UCB OF REQUESTING TERMINAL
Ø
UCB OF REQUESTING TERMINAL
$MCRCB
POINTER TO NEXT BLOCK
UCB OF REQUESTING TERMINAL
POINTER TO NEXT BLOCK
UCB OF REQUESTING TERMINAL
Ø

- MCR queue entries are allocated in a length of 84. bytes (M$$CRB)
- MCR queue entries are linked FIFO
- Entries are inserted by the terminal driver whenever an unsolicited input is completed or when a line is finished being typed in after an MCR> prompt that follows a control-C.
- Entries are dequeued by the MCR root; internal MCR commands are processed by the corresponding overlay segment; MCR queue entries for external MCR commands are inserted into the following queue

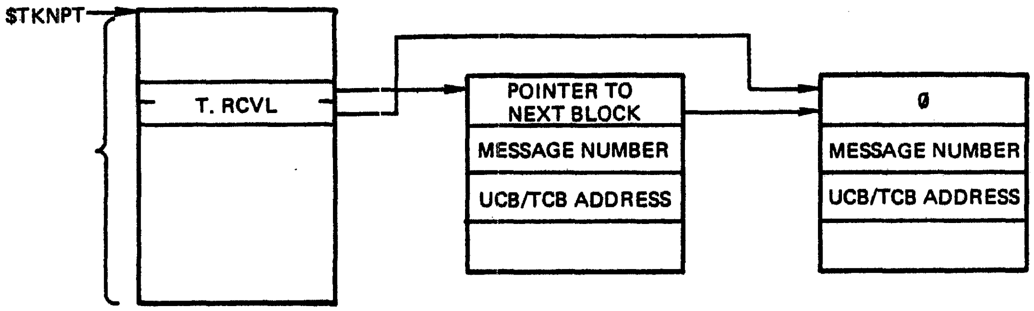
- MCR queue entries in this list are dequeued when an external MCR command (a task with a taskname . . . XYZ) issues a GMCR$ directive; the contents of the entry is copied into the DPB of the GMCR$ directive and the MCR queue entry is deallocated

Figure 8-17 MCR Queues


$PKAVL
POINTER TO FIRST I/O PACKET
$PKNUM
$PKMAX
MAX NO. OF PACKETS ALLOWED IN LIST
NO. OF PACKETS CURRENTLY IN LIST
0

- Each preallocated I/O packet has a length of 44 bytes.
- Preallocated I/O packets are provided - as long as available - whenever $ALOCB is called and a core block with a length of 44 bytes is requested.
- Preallocated I/O packets can and will be used for other purposes also.

Figure 8-18 Pre-allocated I/O Packet Queue



- The queue is linked FIFO
- An entry is made when the routine $DYMSG is called

As a second TKTN queue the active task list (ATL) is used.

There are two types of ATL entries significant for the TKTN:

- TCBs (with the TS.MSG bit in T. STAT and the T2. ABO bit in T. ST2 set) of aborted tasks or an exited task with outstanding I/O in which case

T. EFLG	current I/O count	abort message

- TCBs (with the T2. CAF bit in T. ST2 set and the T3. CAL bit in T. ST3 clear) of tasks with a checkpoint allocation failure

Figure 8-19 Task Termination Notification (TKTN) Queues

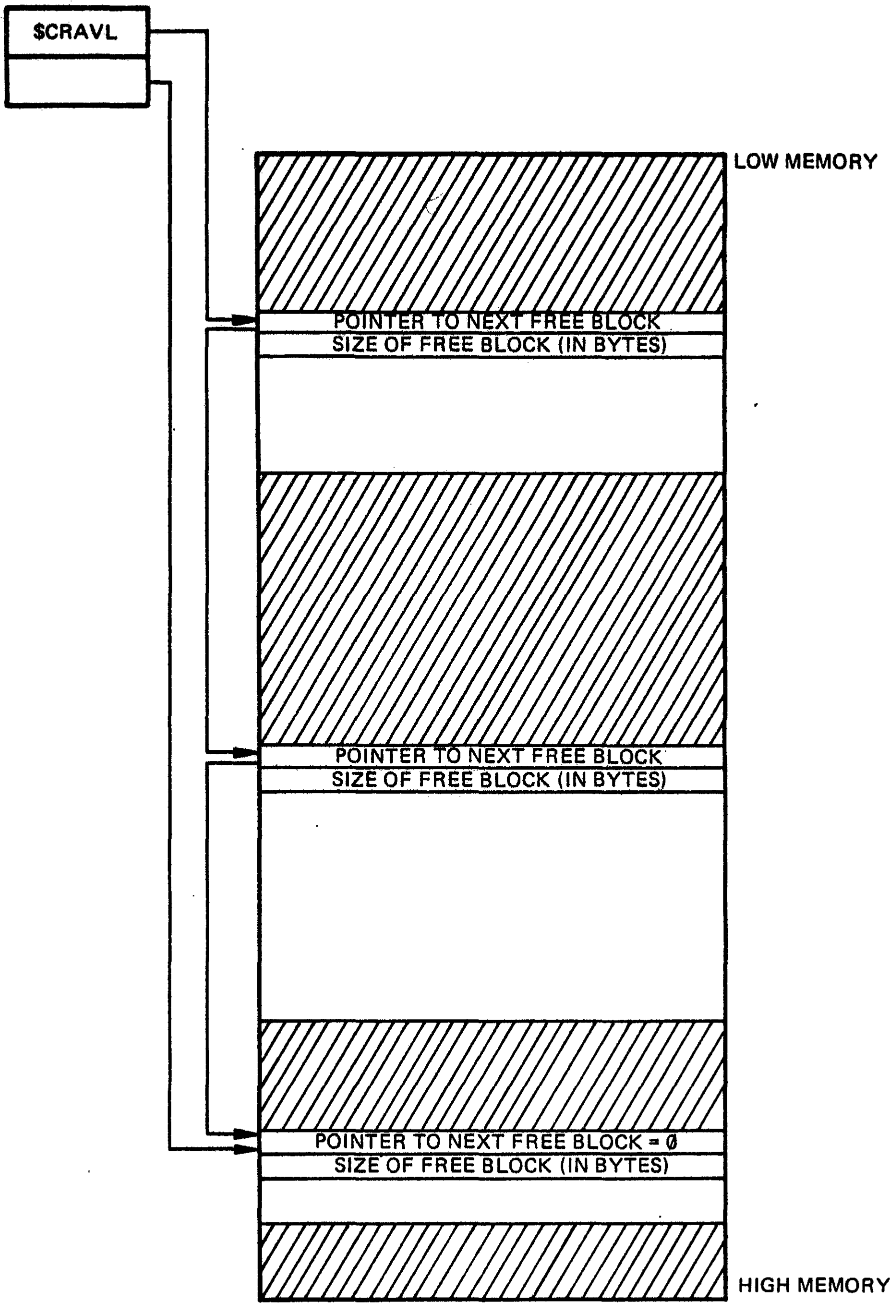

$CRAVL
LOW MEMORY
POINTER TO NEXT FREE BLOCK
SIZE OF FREE BLOCK (IN BYTES)
POINTER TO NEXT FREE BLOCK
SIZE OF FREE BLOCK (IN BYTES)
POINTER TO NEXT FREE BLOCK = Ø
SIZE OF FREE BLOCK (IN BYTES)
HIGH MEMORY

Figure 8-20 Dynamic Storage Region Free Block Queue

8.2.15 Interrelationship Of The DCB, UCB, And SCB

This section discusses the relationships existing among the DCB, UCB, and SCB.

Figure 8-21 shows the data structure resulting from three LA36 DECwriters interfaced by means of a DH11 multiplexer. The structure requires one DCB, three UCBs, and three SCBs, because activity on all three units can proceed in parallel.

Figure 8-22 shows the internal data structure for an RK11 disk controller with three units attached. Note that only one SCB exists because only one of the three units can be active at any given time.

Figure 8-23 shows the data structure for two RK11 disk controllers, each of which has two drives attached. Here, there are two SCBs, because both of the disk controllers can operate in parellel.

Taken together, Figures 8-21, 8-22, and 8-23 illustrate the strategy underlying the existence of three basic I/O control block types. There need be only one DCB for each device type. There may be one or more SCBs, depending on the degree of parallelism that is desired or possible: one for each device-unit, or one for each controller servicing several device-units. The number of UCBs and SCBs, and their interrelationships, are uniquely determined by the hardware that these data structures describe.

This data structure provides considerable flexibility in configuring I/O devices, and, because of the control and status in the structures themselves, substantially reduces the code requirements of the associated drivers.

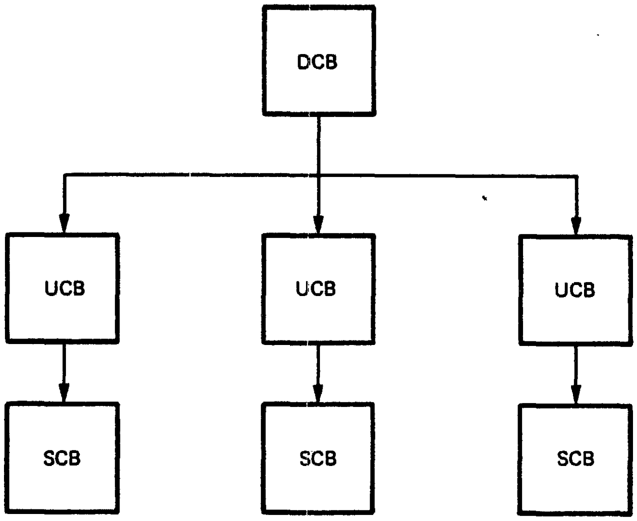


Figure 8-21 DH11 Terminal I/O Data Structure

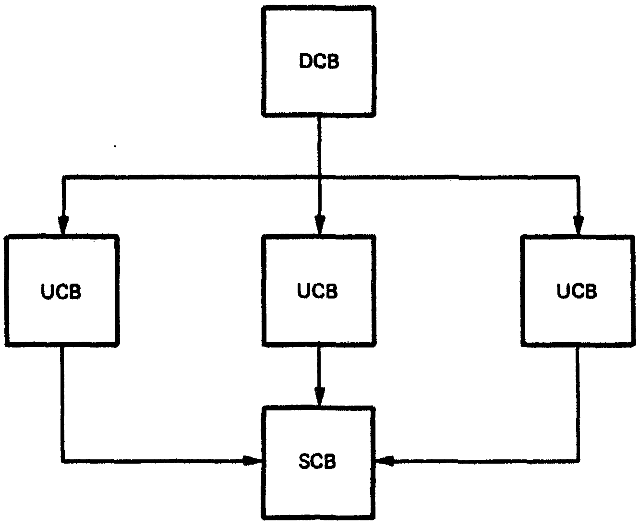


Figure 8-22 RK11 Disk I/O Data Structure

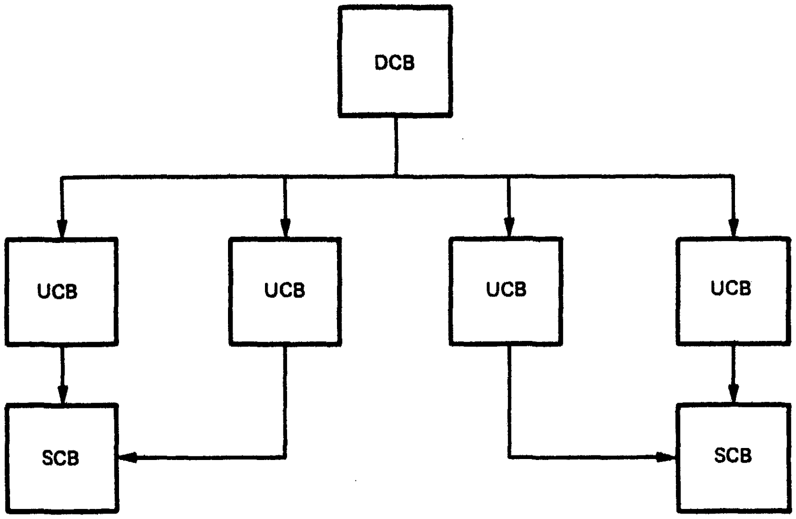


Figure 8-23 I/O Data Structure for Two RK11 Disk Controllers

8.3 I/O CONTROL BLOCK LINKAGES

This section discusses all the I/O control blocks, in terms of their linkage and use within the system. The following data structures make up the complete set for I/O processing:

1. Task Header
2. Window Block (WB)
3. File Control Block (FCB)
4. $DEVHD word, the Device Control Block (DCB), and the Driver Dispatch Table (DDT)
5. Unit Control Block (UCB)
6. Status Control Block (SCB)
7. I/O Queue
8. Volume Control Block (VCB)

Figure 8-24, which provides the structure for the following discussion, shows all the individual data structures and the important link fields within them. The numbers on the figure are keyed to the text to simplify the discussion and to guide the reader through the data structures.

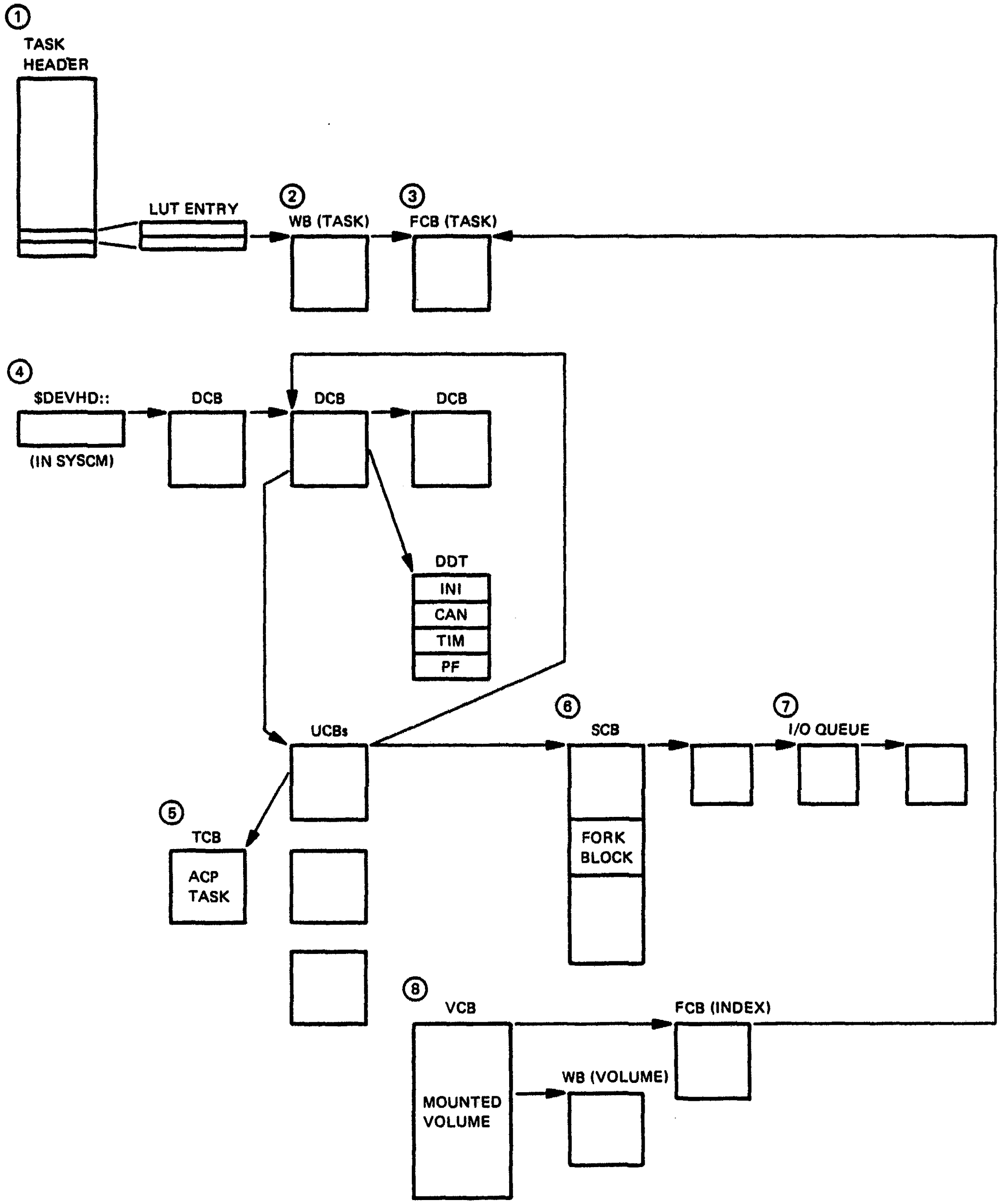


Figure 8-24 I/O Data Structure

1. The task header is constructed during the task-build process. In mapped systems, a copy of the task header (located in the task's partition) is made in the Executive's Dynamic Storage Region. The Executive then uses this copy. To access the current information in this copy, a task must be privileged and mapped to the Executive.

   The task header entry of interest, the Logical Unit Table (LUT), is allocated by the Task Builder and filled in at task installation. The number of LUT entries is established by the UNITS= keyword option; this number is an upper limit on the number of device units a task may have active simultaneously.

   Each LUT entry contains two words: a pointer to an associated UCB, and a pointer to a Window Block. The first word contains the address of the device UCB in the Executive system tables that contains drive-dependent information. The UCB address is set during task installation if a corresponding ASG= option is specified at task build time. The UCB address can also be set at run-time with the Assign LUN directive to the Executive.

   The second word is a pointer to the Window Block if the device is file structured. The Window Block pointer is set when a file is opened on the device whose UCB address is specified by the first word. The Window Block pointer is cleared when the file is closed.

2. A Window Block (WB) exists for each active access to files on a mounted volume. It helps to speed up the retrieval of data items in the file; entries in a WB consist mainly of pointers to contiguous areas on the device on which the file resides. The driver is not concerned with the WB.

3. Each uniquely opened file on a mounted volume has an associated File Control Block (FCB). The file system uses the FCB to control access to the file.

4. $DEVHD is a word located in system common (SYSCM) and points to a singly linked, unidirectional list of Device Control Blocks (DCBs). Each device type in a system has at least one associated DCB. The DCB list is terminated by a 0 in the link word.

   Linked to each DCB is a Driver Dispatch Table (DDT), which is part of the driver. The DDT contains the addresses of the driver's four entry points that the Executive can call.

5. A key data structure is the Unit Control Block (UCB). All the UCBs associated with a DCB appear in consecutive memory locations. During internal processing of a I/O request, most drivers set R5 to the address of the related UCB; it is by means of pointers in the UCB that the driver can access other control blocks in the data structure. In particular, the UCB contains pointers to the DCB, SCB, VCB, and to the UCB to which it may have been redirected. If a Redirect command has not been issued for the device-unit, the UCB redirect pointer points to the UCB itself. When servicing a request for one of its UCBs, the driver is unaware of whether I/O was issued directly to its own UCB or to a UCB that had been redirected to its UCB.

6. One Status Control Block (SCB) exists for each controller in the system. A unique SCB must exist for each controller/device-unit capable of performing parallel I/O. The SCB contains the fork-block storage required when a driver calls $FORK to transfer itself to the fork processing level. The I/O request queue listhead is also contained in the SCB. Generally, register R4 contains the address of the SCB during processing of an I/O request.

7. The I/O queue is a list of control blocks called I/O packets that QIO processing creates dynamically. Each time a task makes an I/O request, the Executive performs a series of validity checks on the DPB parameters. If these checks prove successful, the Executive generates a data structure called an I/O packet. The Executive then inserts the packet into a device-specific, priority-ordered list of packets called the I/O queue. Each I/O queue's listhead is in the SCB to which the I/O requests apply.

   When a device driver needs work, it requests the Executive to dequeue the next I/O packet and deliver it to the requesting driver. Normally, the driver does not directly manipulate the I/O queue. However, an exception is the case where a driver examines an I/O packet before it is queued, or in place of having it queued. For the driver to accomplish this examination, it must set the UC.QUE bit in the control byte, U.CTL, of the UCB.

   The most common reason for a driver to examine a packet before queuing is that the driver employs a special user buffer, other than the normal buffer used in a transfer request. Within the context of the requesting task, the driver must address-check and relocate such a special buffer.

8. One Volume Control Block (VCB) exists for each mounted volume in the system. The VCB maintains volume-dependent control information. It contains pointers to the File Control Block (FCB) and Window Block (WB), which control access to the volume's index file. (The index file is a file of file headers.) The WB for the index file serves the same function as the WB for a user file. All unique FCBs for a volume form a linked list emanating from the index file FCB. This linkage helps to keep file access times short. Further, because the window that contains the mapping pointers is variable in length, you can increase file access speed by adding more access pointers to whatever extent the application requires. However, greater speed requires more main memory.

8.4 CONTROL BLOCK OFFSET DEFINITIONS

8.4.1 Asynchronous System Trap Control Block (ASTCB)

Defined by: .MACRO PKTDF$,L,B,SYSDEF

Some positional dependencies between the DCB and the AST control block are relied upon in the routine $FINXT in the module SYSXT.

```
        .ASECT
.=177774
A.KSR5:'L'  .BLKW 1       ;Subroutine KISAR5 bias (A.CBL=0)
A.DQSR:'L'  .BLKW 1       ;Dequeue subroutine address (A.CBL=0)
            .BLKW 1       ;AST queue thread word
A.CBL:'L'   .BLKW 1       ;Length of control block in bytes
A.BYT:'L'   .BLKW 1       ;Number of bytes to allocate on task stack
A.AST:'L'   .BLKW 1       ;AST trap address
A.NPR:'L'   .BLKW 1       ;Number of AST parameters
A.PRM:'L'   .BLKW 1       ;First AST parameter
```

8.4.1.1 **I/O Packet Offset Definitions** - QIO directive processing constructs the 18-word I/O packet and places it in the driver I/O queue. A call to $GTPKT subsequently delivers the packet to the driver.

```
        .ASECT
.=0
I.LNK:'L'   .BLKW 1       ;I/O queue thread word
                            Links I/O packets queued for a driver.  A
                            0 ends the chain.  The listhead is in the
                            SCB (S.LHD).
I.PRI:'L'   .BLKB 1       ;Request priority
                            Priority copied from the TCB of the re-
                            questing task.
I.EFN:'L'   .BLKB 1       ;Event flag number
                            Contains the event flag number as copied
                            by QIO directive processing from the reques-
                            tor's DPB.
I.TCB:'L'   .BLKW 1       ;TCB address of the requesting task
I.LN2:'L'   .BLKW 1       ;Pointer to second LUN word
                            Contains the address of the second word of
                            the LUT entry in the task header to which
                            the I/O request is directed.  For open
                            files on file-structured devices, this word
                            contains the address of the window block,
                            otherwise it is 0.

I.UCB:'L'   .BLKW 1       ;Pointer to unit control block that
                            contains the address of the redirect UCB
                            if the starting UCB has been subject to a
                            Redirect command.
I.FCN:'L'   .BLKW 1       ;I/O function code
                            Contains the function code for the I/O
                            request.
I.IOSB:'L'  .BLKW 1       ;Virtual address of I/O status block
            .BLKW 1       ;I/O status block relocaton bias
            .BLKW 1       ;I/O status block address
                            I.IOSB contains the virtual address of the
                            I/O status block (IOSB) if one is specified
                            or 0 if one is not.
                              I.IOSB+2 and I.IOSB+4 contain the address
                            doubleword for the IOSB.  In an unmapped
                            system, the first word is 0 and the se-
                            cond word is the real address of the IOSB.
                            In a mapped system, the first word contains
                            the relocation bias of the IOSB; the bias is
                            the 32-word block number in which the IOSB
                            starts.  This block number is derived by
                            viewing available real memory as a col-
                            lection of 32-word blocks numbered consecu-
```

```
                          tively, starting with 0. Thus, if the
                          IOSB starts at physical location 3210(8),
                          its block number is 32(8). See the RSX-11M
                          Guide to Writing an I/O Driver for more
                          details.
I.AST:'L'   .BLKW 1     ;AST service routine address
                          Contains the virtual address of the AST
                          service routine to be executed at I/O com-
                          pletion.  If no address is specified, the
                          field contains 0.
I.PRM:'L'   .BLKW 1     ;Reserved for mapping parameter #1
            .BLKW 6     ;Parameters 1 to 6 constructed from the last
                          6 words of the DPB.  Note that if the I/O
                          function is a transfer, the buffer address
                          (the first DPB device-dependent paramenter)
                          is translated to an equivalent address
                          doubleword.  Therefore, device-dependent
                          paramenter "n" is copied to I.PRM+(2*n)+2.
            .BLKW 1     ;User mode diagnostic parameter word
I.ATTL='B'.             ;Minimum length of I/O packet (used by
                          file system to calculate maximum
                          number of attributes)
I.LGTH='B'.             ;Length of I/O request control block
```

8.4.2 Clock Queue Control Block (CQCB)

Defined by: .MACRO CLKDF$,L,B

There are five types of clock queue control blocks. Each control block has the same format in the first five words and differs in the remaining three.

The following control block types are defined:

```
C.MRKT='B'0             ;Mark time request
C.SCHD='B'2             ;Task request with periodic rescheduling
C.SSHT='B'4             ;Single shot task request
C.SYST='B'6             ;Single shot internal system subroutine
                          (ident)
C.SYTK='B'8.            ;Single shot internal system subroutine (task)
C.CSTP='B'10.           ;Clear stop bit (conditionalized on shuffling)
```

8.4.2.1 Clock Queue Control Block Independent Offsets

```
     .ASECT
.=0
C.LNK:'L'     .BLKW 1   ;Clock queue thread word
C.RQT:'L'     .BLKB 1   ;Request type
C.EFN:'L'     .BLKB 1   ;Event flag number (mark time only)
C.TCB:'L'     .BLKW 1   ;TCB address or system subroutine
                        ; identification
C.TIM:'L'     .BLKW 2   ;Absolute time when request comes due
```

8.4.2.2 Clock Queue Control Block - Mark Time Dependent Offsets

```
.=C.TIM+4                   ;Start of dependent area
C.AST:'L'      .BLKW 1      ;AST address
C.SRC:'L'      .BLKW 1      ;Flag mask word for 'BIS'  source
C.DST:'L'      .BLKW 1      ;Address of 'BIS' destination
```

8.4.2.3 Clock Queue Control Block - Periodic Rescheduling Dependent Offsets

```
.=C.TIM+4                   ;Start of dependent area
C.RSI:'L'      .BLKW 2      ;Reschedule interval in clock ticks
C.UIC:'L'      .BLKW 1      ;Scheduling UIC
```

8.4.2.4 Clock Queue Control Block - Single-Shot Dependent Offsets

```
.=C.TIM+4                   ;Start of dependent area
               .BLKW 2      ;Two unused words
               .BLKW 1      ;Scheduling UIC
```

8.4.2.5 Clock Queue Control Block - Single-Shot Internal Subroutine Offsets

There are two type codes for this type of request:

Type 6: single shot internal subroutine with a 16-bit value as an identifier.

Type 8: single shot internal subroutine with a TCB address as an identifier.

```
.=C.TIM+4                   ;Start of dependent area
C.SUB:'L'      .BLKW 1      ;Subroutine address
C.AR5:'L'      .BLKW 1      ;Relocation base (for loadable drivers)
               .BLKW 1      ;One unused word
C.LGTH='B'.                 ;Length of clock queue control block
```

8.4.3 Communications Control Block (CCB)

Defined by: .MACRO CUCDF$ X,Y

```
     .ASECT
.=U.CW2+2                   ;Position ACU at U.CW3 position
U.ACUR:'X      .BLKW 1      ;ACU register address (at U.CW3)
U.NSYN:'X      .BLKB 1      ;Number of syncs to issue
U.NSYC:'X      .BLKB 1      ;Number of syncs issued

.=U.VCB+2                   ;Extend after VCB address
U.PHDR:'X      .BLKW 1      ;End-of-transmit header check routine
U.RCHK:'X      .BLKW 1      ;Address of redundancy check routine
U.QSYN:'X                   ;DQ sync buffer
U.RCAC:'X      .BLKW 1      ;Redundancy check accumulator
U.RBUF:'X      .BLKW 2      ;Receive buffer address
U.RCNT:'X      .BLKW 1      ;Receive buffer count
U.SVC:'X       .BLKW 1      ;External receive interrupt service
U.TXCT:'X                   ;DQ transmit buffer count
U.INTP:'X      .BLKW 1      ;Internal receive interrupt service
U.SYNC:'X      .BLKB 1      ;Current sync character
U.MPN:'X       .BLKB 1      ;Multipoint node number
```

```
U.RFRK:'X     .BLKW 1   ;Receive fork block
U.RFPC:'X     .BLKW 1   ;Fork PC
U.RFR5:'X     .BLKW 1   ;Fork save R5
U.RFR4:'X     .BLKW 1   ;Fork save R4
U.RFCT='Y     U.TXCT+1  ;DQ receive fork count
     .PSECT

U2.HDX='Y     100000    ;Set if running half duplex
U2.LIN='Y      40000    ;Set if physical link half duplex
U2.CTS='Y      20000    ;Set if clear to send expected
U2.SWC='Y      10000    ;Set if switched circuit
U2.ONL='Y       4000    ;Set if unit on line
U2.HPT='Y       1000    ;Set if DQ has protocol option
U2.HRC='Y        400    ;Set if controller does CRC
U2.RCV='Y        200    ;Set if receiver active, half duplex
U2.ACU='Y        100    ;Set if unit has ACU
U2.MPT='Y         40    ;Set if operating multipoint
U2.FTM='Y         20    ;Set if just initiated receive
U2.SFL='Y         10    ;Sent final on last message
U2.RFK='Y          4    ;Clear when primary fork free (DQ)
U2.SYC='Y          3    ;Bits used for receiver sync count
U2.TXA='Y          2    ;Set if TX code active (DQ)
U2.SNC='Y          1    ;Set if sending syncs (DQ)
US.SYN='Y          1    ;Set if device always sends sync

U3.LOK='Y     100000    ;Software interrupt lockout (DA)
U3.RPD='Y        400    ;Set when receive pending (DA)
U3.RAC='Y        200    ;Set when receive in progress (DA)
U3.SND='Y          1    ;Set when in transmit mode (DA)
              .MACRO CUCDF$ A,B
              .ENDM
              .ENDM
```

8.4.3.1 **Communications Vector**

Defined by: .MACRO CVCDF$ X,Y

```
     .ASECT
.=V.IFWI                ;Overlay from here on
V.CPRT:'X     .BLKW 1   ;Protocol descriptor vector address
V.CLUN:'X     .BLKB 1   ;CCP'S current LUN for this node
V.CMPN:'X     .BLKB 1   ;Multipoint node designator
V.CSTS:'X     .BLKW 1   ;Protocol-oriented status
V.CST1:'X     .BLKB 1   ;Status extension
V.CNID:'X     .BLKB 1   ;Node ID number for next node over
V.CUCB:'X     .BLKW 1   ;Current UCB  for this VCB
V.CMPL:'X     .BLKW 1   ;Multipoint table link word
V.CNPN:'X     .BLKW 1   ;Next multipoint node in chain
V.CMBC:'X     .BLKW 1   ;Multipoint half duplex count of bytes
V.CRED:'X     .BLKW 1   ;Redundancy check keyword
V.CNHD:'X     .BLKW 1   ;Address of (new) header block
V.CTBL:'X     .BLKW 1   ;Address of current AST block
V.CRBL:'X     .BLKW 1   ;Address of current data block
V.NXHD:'X     .BLKW 1   ;Next header buffer address
V.CDBC:'X     .BLKW 1   ;Receiver data count, bytes
V.CIAD:'X     .BLKW 1   ;Receive buffer address, enter transparency
V.CICT:'X     .BLKW 1   ;Receive data count, enter T.
V.CSAD:'X     .BLKW 1   ;Receive buffer address, exit T.
V.CSCT:'X     .BLKW 1   ;Receive buffer count, exit T.
V.CMXI:'X     .BLKW 1   ;Max record size accepted, bytes
V.CMXO:'X     .BLKW 1   ;Max record size can transmit
V.CTDH:'X     .BLKW 1   ;Transmit data packet list head
```

```
V.CTDT:'X      .BLKW 1    ;Transmit data packet list tail
V.CSTQ:'X      .BLKW 2    ;Start mode pre-transmission queue
V.CLMT:'X      .BLKB 1    ;Last message number transmitted
V.CTMA:'X      .BLKB 1    ;Last transmitted message ACKed
V.CLMR:'X      .BLKB 1    ;Last message number received
V.CRMA:'X      .BLKB 1    ;Last received message ACKed
V.CMCT:'X      .BLKW 2    ;Count of messages sent
V.CRPC:'X      .BLKB 1    ;Consecutive rep count
V.CRPL:'X      .BLKB 1    ;Consecutive rep limit
V.CRTM:'X      .BLKW 1    ;Rep timeout value, ticks
V.CREP:'X      .BLKW 1    ;Rep count accumulator
V.CBCC:'X      .BLKW 1    ;Header block check error count
V.CDCC:'X      .BLKW 1    ;Data block check error count
V.CURH:'X      .BLKW 1    ;Unrecognized header error count
V.CDLE:'X      .BLKW 1    ;Data late error count
V.COVR:'X      .BLKW 1    ;Transmitter overrun count
V.CRST:'X      .BLKW 1    ;Number of starts or restarts
V.CALF:'X      .BLKW 1    ;Number of system buffer allocation failures
V.CREA:'X      .BLKB 1    ;Contains NAK reason
               .BLKB 1    ;Unused

               .IF DF C$$ACU
V.CTPN:'X      .BLKB 12. ;Telephone number, this unit (ASCII)
               .ENDC

V.CLEN:'X      .BLKW 1    ;Communications VCB length

     .PSECT
VS.SEL='Y      100000     ;Set if select other unit
VS.FIN='Y       40000     ;Set if final transmission
VS.REP='Y       20000     ;Unit in REP mode
VS.RTR='Y       10000     ;Transmitter in retry mode
VS.STK='Y        4000     ;Set if start ACK expected
VS.STR='Y        2000     ;Start mode if set
VS.ACK='Y        1000     ;Set if should send ACK
VS.NAK='Y         400     ;Set if should send NAK
VS.MPT='Y         200     ;Set if link is multipoint
VS.DIS='Y         200     ;(V.CST1) Set link in dismount mode
VS.CTL='Y         100     ;Set if have mastership, multipoint
VS.DAT='Y          40     ;Set if data message
VS.SYN='Y          20     ;Set if sync train required
VS.POL='Y          10     ;Poll next multipoint if set
VS.ONL='Y           4     ;Related node is active
VS.BOO='Y           2     ;Marked as boot channel
VS.TRN='Y           1     ;Marked as DDCMP-only channel
               .MACRO CVCDF$ A,B
               .ENDM
               .ENDM
```

8.4.4 Device Control Block (DCB)

Defined by: .MACRO DCBDF$,L,B

The Device Control Block (DCB) describes the static characteristics (rather than execution-time information) of both the device controller and the units attached to the controller. Most of the data in the DCB is established in the assembly source for the I/O driver data structure. The DCB is used by the QIO directive processing code in the Executive, rather than by the driver. All the DCBs in a system form a forward-linked list, with the last DCB having a link of 0. The link word is D.LNK.

At least one DCB exists for each type of device appearing in a system (device type should not be equated with device-unit). Therefore, a DCB has the device name in it; for example, the DCB for DK0: has ".ASCII /DK/" in it. In general, there is only one DCB containing the name of a given device. However, in the case of the terminal driver, a separate DCB exists for every kind of terminal device controller in the system (DL, DJ, DH, and DZ), and each DCB contains ".ASCII /TT/". For example, if there are terminals in a system, then there is at least one controller and one DCB containing the device name "TT". If some of the terminals were interfaced via DL11-As and the rest via a DH11, then there would be two DCBs; one for all DL11-A-interfaced terminals, and one for all DH11-interfaced terminals. They both would contain the device name, "TT".

```
              .ASECT
.=0
D.LNK:'L'     .BLKW 1    ;Link to next DCB.  A 0 in this field in-
                           dicates the last (or only) DCB in the chain.
                           The driver links its DCB into the system
                           DCBs via the global label $USRTB on its
                           first DCB.
D.UCB:'L'     .BLKW 1    ;Pointer to first Unit Control Block. All
                           UCBs, for a given DCB, are in contiguous
                           memory locations and must all be the same
                           length.
D.NAM:'L'     .BLKW 1    ;Generic device name in ASCII by which
                           device units are referenced mnemonically.
D.UNIT:'L'    .BLKB 1    ;Lowest unit number covered by this DCB.
              .BLKB 1    ;Highest unit number covered by this DCB.
                           These two bytes represent the range of
                           logical units available to the user for
                           assignment.  Typically, the lowest number is
                           0 and the highest is n-1, where n is the
                           number of device-units described by the DCB.
D.UCBL:'L'    .BLKW 1    ;Length of each unit control block in
                           bytes.  The UCB can have any length to meet
                           the needs of the driver for variable
                           storage.  However, all UCBs for a given DCB
                           must have the same length.  The specified
                           length must include prefix words (U.LUIC and
                           U.OWN) if present.

D.DSP:'l'     .BLKW 1    ;Pointer to driver dispatch table.
                           When the Executive must enter the driver
                           at any of the four entry points contained
                           in the driver dispatch table, it accesses
                           D.DSP, locates the address in the table, and
                           calls the driver at that address. A 0
                           table address indicates that the (loadable)
                           driver is not in memory.  If a driver does
                           not process a given function, it supplies a
                           return address.  For more details see the
                           RSX-11M Guide to Writing an I/O Driver.
D.MSK:'l'     .BLKW 1    ;Legal function mask  codes 0-15.
              .BLKW 1    ;Control function mask codes 0-15.
              .BLKW 1    ;NOP'ed function mask codes 0-15.
              .BLKW 1    ;ACP function mask codes 0-15.
              .BLKW 1    ;Legal function mask codes 16.-31.
              .BLKW 1    ;Control function mask codes 16.-31.
              .BLKW 1    ;NOP'ed function mask codes 16.-31.
              .BLKW 1    ;ACP function mask codes 16.-31.
                           The Executive uses these words to validate
                           and dispatch the I/O request specified
                           by a QIO directive.
```

```
D.PCB:'L'     .BLKW 1   ;Address of the driver's Partition Control
                          Block (PCB).  This word is present in
                          the DCB only if the loadable-driver
                          SYSGEN option has been selected.  It must
                          be initialized to 0.  The DCB can be
                          extended by adding words after D.PCB.
                            A PCB exists for every partition in a
                          system.  MCR creates a PCB when the SET
                          /MAIN or SET /SUB commands are given.  If a
                          driver is loadable, its PCB describes the
                          partition in which it resides.
                            The Executive uses D.PCB together with
                          D.DSP (the address of the driver dispatch
                          table) to determine whether a driver is
                          loadable or resident, and, if loadable,
                          whether or not it is in memory.  For more
                          details see the RSX-11M Guide to Writing an
                          I/O Driver.
```

8.4.4.1 Driver Dispatch Table Offsets

```
D.VINI='B'0             ;Device initiator
D.VCAN='B'2             ;Cancel current I/O function
D.VOUT='B'4             ;Device timeout
D.VPWF='B'6             ;Powerfail recovery
```

8.4.5 Error Message Block (EMB)

Defined by: .MACRO EMBDF$,L,B

The following error codes are defined:

```
EC.INI='B'40
EC.DVC='B'1             ;Device error bit set
EC.DTO='B'140           ;Device interrupt timeout
EC.NSI='B'141           ;Undefined interrupts
EC.LOA='B'4             ;Driver load
EC.UNL='B'10            ;Driver unload
EC.MPE='B'2             ;Memory parity error
EC.PWR='B'42            ;Power fail
```

8.4.5.1 Error Message Block Independent Offsets

```
      .ASECT
.=0
E.SIZE:'L'    .BLKW 1   ;Size of the EMB in bytes
E.CODE:'L'    .BLKB 1   ;Error code
              .BLKB 1   ;Reserved
E.TIME:'L'    .BLKB 6   ;Time of error-s,m,h,d,m,y
E.SEQ:'L'     .BLKW 1   ;Sequence number
E.ABM:'L'     .BLKW 1   ;Saved I/O active bitmap
```

8.4.5.2 Error Message Block - Undefined Interrupt Offsets

```
.=E.ABM+2
E.VCTR:'L'    .BLKB 1   ;ID of vector trapped through
E.LOST:'L'    .BLKB 1   ;Number of lost undefined interrupts
E.OPS:'L'     .BLKW 1   ;Old PS preinterrupt
E.OPC:'L'     .BLKW 1   ;Old PC preinterrupt
```

8.4.5.3 Error Message Block - Device Errors and Device Interrupt Timeout Offsets

```
.=E.ABM+2
E.RTRY:'L'    .BLKB 2   ;Low byte=left, hi byte=start
              .BLKB 1   ;Task priority
E.IOC:'L'     .BLKB 1   ;I/O in progress count
E.TASK:'L'    .BLKW 2   ;Task name
E.PAR:'L'     .BLKW 1   ;Partition address
E.UIC:'L'     .BLKW 1   ;Task UIC
E.UCB:'L'     .BLKW 1   ;Device UCB address
E.FCN:'L'     .BLKB 2   ;QIO function code
E.PRM:'L'     .BLKW 1   ;Buffer mapping
              .BLKW 6   ;Parameters 1-6
E.RCNT:'L'    .BLKB 1   ;Number of device registers
              .BLKB 1   ;Reserved
E.REGS='B'.
E.LGTH='B'.
```

8.4.5.4 Error Message Block - Driver Load and Unload

```
.=E.TIME+<9.*2>
E.WHY:'L'     .BLKW 1   ;Action code (load/unload)
.=.+2
E.NAME:'L'    .BLKW 1   ;Device name (ASCII)
```

8.4.6 File Control Block (FCB)

```
              .ASECT
.=0
F.LINK:       .BLKW 1   ;FCB chain pointer
F.FNUM:       .BLKW 1   ;File number
F.FSEQ:       .BLKW 1   ;File sequence number
              .BLKB 1   ;Not used
F.FSQN:       .BLKB 1   ;File segment number
F.FOWN:       .BLKW 1   ;File owner's UIC
F.FPRO:       .BLKW 1   ;File protection code
F.UCHA:       .BLKB 1   ;User controlled characteristics
F.SCHA:       .BLKB 1   ;System controlled characteristics
F.HDLB:       .BLKW 2   ;File header logical block number
F.LBN:        .BLKW 2   ;LBN of virtual block 1 if contiguous
                        ;0 if non contiguous
F.SIZE:       .BLKW 2   ;Size of file in blocks
F.NACS:       .BLKB 1   ;No. of accesses
F.NLCK:       .BLKB 1   ;No. of locks
F.STAT:                 ;FCB status word
F.NWAC:       .BLKB 1   ;Number of write accessors
              .BLKB 1   ;Status bits for FCB consisting of
     FC.WAC=100000      ;Set if file accessed for write
     FC.DIR=40000       ;Set if FCB is in directory LRU
     FC.CEF=20000        ;Set if directory EOF needs updating
     FC.FCO=10000        ;Set if trying to force directory contig
F.DREF:       .BLKW 1    ;Directory EOF block number
F.DRNM:       .BLKW 1    ;1st word of directory name
F.FEXT:       .BLKW 1    ;Pointer to extension FCB
F.FVBN:       .BLKW 2    ;Starting VBN of this file segment
F.LKL:        .BLKW 1    ;Pointer to locked block list for file
F.LGTH:                  ;Size in bytes of FCB
```

8.4.6.1 Window

```
     .ASECT
.=0
W.CTL:        .BLKW 1    ;Low byte = # of map entries active
                         ;High byte consists of the following bits
     WI.RDV=400          ; Read virtual block allowed if set
     WI.WRV=1000         ; Write virtual block allowed if set
     WI.EXT=2000         ; Extend allowed if set
     WI.LCK=4000         ; Set if locked against shared access
     WI.DLK=10000        ; Set if deaccess lock enabled
     WI.EXL=40000        ; Set if manual unlock desired
     WI.BPS=100000       ; Bypass access interlock if set
W.VBN:        .BLKB 1    ;High byte of 1st VBN mapped by window
W.WISZ:       .BLKB 1    ;Size in rtrv ptrs of window (7 bits)
              .BLKW 1    ;Low order word of 1st VBN mapped
W.FCB:        .BLKW 1    ;File control block address
W.LKL:        .BLKW 1    ;Pointer to list of users locked blocks
W.RTRV:                  ;Offset to 1st retrieval pointer in window
```

8.4.6.2 Locked Block List Node

```
     .ASECT
.=0
L.LNK:        .BLKW 1    ;Link to next node in list
L.WI1:        .BLKW 1    ;Pointer to window for first entry
L.VB1:        .BLKB 1    ;High order VBN byte
L.CNT:        .BLKB 1    ;Count for entry
              .BLKW 1
L.LGTH:
```

8.4.7 Get Command Line Control Block (GCML)

Defined by: .MACRO GCML$D GBL

```
G.ERR = S.FDB                   ;Error return code byte
G.MODE = G.ERR+1                ;Status and mode control byte
G.PSDS = G.ERR+2                ;Prompt string descriptor
G.CMLD = G.ERR+6                ;Command line descriptor
G.ISIZ = 16.                    ;Size of impure area (PTRS, FLAGS,
                                ; COUNTS, etc.)
G.DPRM = G.ERR+G.ISIZ           ;Default prompt string
G.SIZE = G.DPRM+6+S.FNB         ;Buffer size
```

G.MODE BIT DEFINITIONS

```
GE.COM = 1                    ;Comment recognition
GE.IND = 2                    ;Indirect file recognition
GE.CLO = 4                    ;Close command file before return
GE.LC = 10                    ;Pass lower case characters
GE.CON = 20                   ;Continuation lines allowed
GE.SIZ = 40                   ;Buffer size
```

G.ERR VALUES

```
GE.IOR = -1                   ;I/O error
GE.OPR = -2                   ;Unable to open indirect file
GE.BIF = -3                   ;Bad indirect file name
GE.MDE = -4                   ;Maximum indirect file depth exceeded
GE.EOF = -10.                 ;End of file
GE.RBG = -40.                 ;Buffer size error (overflow)
```

8.4.8 Hardware Definitions

Defined by: .MACRO HWDDF$,L,B

8.4.8.1 Hardware Register Addresses and Status Codes

```
MPCSR='B'177746         ;Address of PDP-11/70 memory parity register
MPAR='B'172100          ;Address of first memory parity register
PIRQ='B'177772          ;Programmed interrupt request register
PR0='B'0                ;Processor priority 0
PR1='B'40               ;Processor priority 1
PR4='B'200              ;Processor priority 4
PR5='B'240              ;Processor priority 5
PR6='B'300              ;Processor priority 6
PR7='B'340              ;Processor priority 7
PS='B'177776            ;Processor status word
SWR='B'177570           ;Console switch and display register
TPS='B'177564           ;Console terminal printer and status register
```

8.4.8.2 Extended Arithmetic Element Registers

```
              .IF DF E$$EAE
AC='B'177302            ;Accumulator
MQ='B'177304            ;Multiplier-quotient
SC='B'177310            ;Shift count
              .ENDC
```

8.4.8.3 Memory Management Hardware Register and Status Codes

```
             .IF DF M$$MGE
KDSAR0='B'172360        ;Kernel D PAR 0
KDSDR0='B'172320        ;Kernel D PDR 0
KISAR0='B'172340        ;Kernel I PAR 0
KISAR5='B'172352        ;Kernel I PAR 5
KISAR6='B'172354        ;Kernel I PAR 6
KISAR7='B'172356        ;Kernel I PAR 7
KISDR0='B'172300        ;Kernel I PDR 0
KISDR6='B'172314        ;Kernel I PDR 6
```

```
KISDR7='B'172316        ;Kernel I PAR 7
SISDR0='B'172200        ;Supervisor I PDR 0
UDSAR0='B'177660        ;User D PAR 0
UDSDR0='B'177620        ;User D PDR 0
UISAR0='B'177640        ;User I PAR 0
UISAR4='B'177650        ;User I PAR 4
UISAR5='B'177652        ;User I PAR 5
UISAR6='B'177654        ;User I PAR 6
UISAR7='B'177656        ;User I PAR 7
UISDR0='B'177600        ;User I PDR 0
UISDR4='B'177610        ;User I PDR 4
UISDR5='B'177612        ;User I PDR 5
UISDR6='B'177614        ;User I PDR 6
UISDR7='B'177616        ;User I PDR 7


UBMPR='B'170200         ;UNIBUS mapping register 0
CMODE='B'140000         ;Current mode field of PS word
PMODE='B'30000          ;Previous mode field of PS word
SR0='B'177572           ;Segment status register 0
SR3='B'172516           ;Segment status register 3
              .ENDC
```

8.4.8.4 Feature Symbol Definitions

```
FE.EXT='B'1             ;11/70 extended memory support
FE.MUP='B'2             ;Multi-user protection support
FE.EXV='B'4             ;Executive is supported to 20K
FE.DRV='B'10            ;Loadable driver support
FE.PLA='B'20            ;PLAS support
FE.CAL='B'40            ;Dynamic checkpoint space allocation
FE.PKT='B'100           ;Preallocation of I/O packets
FE.EXP='B'200           ;Extend Task directive supported
FE.LSI='B'400           ;Processor is an LSI-11
FE.CEX='B'20000         ;COM exec is loaded
FE.MXT='B'40000         ;MCR exit after each command mode
FE.NLG='B'100000        ;Logins disabled - multi-user support
              .ENDM
              .ENDM
```

8.4.9 Interrupt Transfer Block (ITB)

Defined by: .MACRO ITBDF$ L,B,SYSDEF

```
     .IF DF A$$TRP
     .MCALL  PKTDF$
     PKTDF$             ;Define AST block offsets
     .ENDC

     .ASECT
.=0
X.LNK:'L'     .BLKW 1   ;Link word for ITB lists starting in TCB
X.JSR:'L'     JSR R5,@#0 ;Call $INTSC
X.PSW:'L'     .BLKB 1   ;Low byte of PSW for ISR
              .BLKB 1   ;Unused
X.ISR:'L'     .BLKW 1   ;ISR entry point (APR5 mapping)
X.FORK:'L'              ;Fork block
              .BLKW 1   ;Thread word
              .BLKW 1   ;Fork PC
              .BLKW 1   ;Saved R5
```

```
                .BLKW 1   ;Saved R4
                .IF DF M$$MGE
X.REL:'L'       .BLKW 1   ;Relocation base for APR5
                .ENDC

X.DSI:'L'       .BLKW 1   ;Address of DIS.INT. routine
X.TCB:'L'       .BLKW 1   ;TCB address of owning task
                .IF NB SYSDEF
                .IF DF A$$TRP
                .BLKW 1   ;A.DQSR for AST block
X.AST:'l'       .BLKB A.PRM ;AST block
                .ENDC

X.VEC:'L'       .BLKW 1   ;Vector address (if AST support,
                          ; this is first and only AST parameter)
X.VPC:'L'       .BLKW 1   ;Saved vector PC
X.LEN:'L'                 ;Length in bytes of ITB
                .ENDC
               .PSECT
```

8.4.10 Logical Assignment Control Block

Defined by: .MACRO LCBDF$,L,B

The logical assignment control block (LCB) associates a logical name with a physical device-unit. LCBs are linked together to form the logical assignments of a system. Assignments may be on a system-wide or local (terminal) basis.

```
       .ASECT
.=0
L.LNK:'L'       .BLKW 1   ;Link to next LCB
L.NAM:'L'       .BLKW 1   ;Logical name of device
L.UNIT:'L'      .BLKB 1   ;Logical unit number
L.TYPE:'L'      .BLKB 1   ;Type of entry (0=system wide)
L.UCB:'L'       .BLKW 1   ;TI UCB address
L.ASG:'L'       .BLKW 1   ;Assignment UCB address
L.LGTH='B'.-L.LNK         ;Length of LCB
      .PSECT
```

8.4.11 Partition Control Block (PCB)

Defined by: .MACRO PCBDF$ L,B,SYSDEF

```
      .ASECT
.=0
P.LNK:'L'       .BLKW 1   ;Link to next PCB. The PCBs are linked in
                            physical address order, highest to lowest.
                            If a main partition has subpartitions, they
                            are linked in the PCB chain off the main
                            partition in highest to lowest address
                            order.  The last subpartition of a main
                            partition either ends the PCB chain or
                            links to the next main partition.  A main
                            partition with no subpartition either links
                            to the next main partition or ends the
                            chain.
P.PRI:'L'       .BLKB 1   ;Priority of partition
P.IOC:'L'       .BLKB 1   ;I/O + I/O status block count
P.NAM:'L'       .BLKW 2   ;Partition name in RAD50
```

```
P.SUB:'L'       .BLKW 1     ;Pointer to next subpartition. Structured and
                              used similarly to P.LNK when manipulating a
                              chain of subpartition PCBs.
P.MAIN:'L'      .BLKW 1     ;Pointer to main partition. The backpointer
                              from a subpartition to its parent main par-
                              tition.
                .IF NB SYSDEF
                .IF NDF M$$MGE
P.HDR:'L'                   ;Pointer to task header (unmapped system)
                .ENDC
                .IFTF
P.REL:'L'       .BLKW 1     ;Starting physical address of the partition.
                              The partition base relocation bias. In a
                              mapped system, P.REL is the bias; in an un-
                              mapped system, P.REL is the actual parti-
                              tion address.
P.BLKS:'L'
P.SIZE:'L'      .BLKW 1     ;Size of partition in bytes
P.WAIT:'L'      .BLKW 2     ;Partition wait queue listhead (2 words). A
                              pointer to a list of tasks awaiting the use
                              of the partition.  The list is ordered by
                              priority and is searched to determine which
                              task should be in control of the partition.
P.SWSZ:'L'      .BLKW 1     ;Partition swap size (system-controlled
                              partitions only)
P.BUSY:'L'      .BLKB 2     ;Partition busy flags.  The first byte, the
                              busy status, is the inclusive OR of the
                              state for the main partition and all its
                              subpartitions.  The second byte, the busy
                              mask, contains a busy (1) or not busy (0)
                              setting for the main partition and its
                              seven subpartitions.
P.OWN:'L'
P.TCB:'L'       .BLKW 1     ;TCB address of the task that owns the par-
                              tition (owner task).
P.STAT:'L'      .BLKW 1     ;Partition status flags
                .IFT
                .IF DF M$$MGE
P.HDR:'L'       .BLKW 1     ;Pointer to task header (mapped system)
                .ENDC

P.PRO:'L'       .BLKW 1     ;Protection word [DEWR,DEWR,DEWR,DEWR]
P.ATT:'L'       .BLKW 2     ;Attachment descriptor listhead

                .IF NDF P$$LAS
P.LGTH='B'P.PRO             ;Length of partition control block
                .IFF
P.LGTH='B'.                 ;Length of partition control block
                .ENDC
                .IFF
     .PSECT
```

8.4.11.1 Partition Status Word Bit Definitions

```
PS.OUT='B'100000        ;Partition is out of memory (1=yes)
PS.CKP='B'40000         ;Partition checkpoint in progress (1=yes)
PS.CKR='B'20000         ;Partition checkpoint is requested
                        ; (1=yes)
PS.CHK='B'10000         ;Partition is not checkpointable (1=yes)
PS.FXD='B'4000          ;Partition is fixed (1=yes)
PS.PER='B'2000          ;Parity error in partition (1=yes)
PS.LIO='B'1000          ;Marked by shuffler for long I/O (1=yes)
PS.NSF='B'400           ;Partition is not shuffleable (1=yes)
PS.COM='B'200           ;Library or common block (1=yes)
PS.PIC='B'100           ;Position independent library or common
                        ; (1=yes)
PS.SYS='B'40            ;System controlled partition (1=yes)
PS.DRV='B'20            ;Driver is loaded in partition (1=yes)
PS.DEL='B'10            ;Partition should be deleted when not
                        ; attached (1=yes)
PS.APR='B'7             ;Starting APR number mask
```

8.4.11.2 Attachment Descriptor Offsets

```
     .ASECT
.=0
A.PCBL:'L'    .BLKW 1   ;PCB attachment queue thread word
A.PRI:'L'     .BLKB 1   ;Priority of attached task
A.IOC:'L'     .BLKB 1   ;I/O count through this descriptor
A.TCB:'L'     .BLKW 1   ;TCB address of attached task
A.TCBL:'L'    .BLKW 1   ;TCB attachment queue thread word
A.STAT:'L'    .BLKB 1   ;Status byte
A.MPCT:'L'    .BLKB 1   ;Mapping count of task thru this
                        ; descriptor
A.PCB:'L'     .BLKW 1   ;PCB address of attached task
A.LGTH='B'.             ;Length of attachment descriptor
```

8.4.11.3 Attachment Descriptor Status Byte Bit Definitions

```
     .PSECT
AS.DEL='B'10            ;Task has delete access (1=yes)
AS.EXT='B'4             ;Task has extend access (1=yes)
AS.WRT='B'2             ;Task has write access (1=yes)
AS.RED='B'1             ;Task has read access (1=yes)
              .ENDC
```

8.4.12 Region Definition Block (RDB)

Defined by: .MACRO RDBDF$ GBL

```
     .MCALL .BLKW.,.BLK.
     .BLK.
     .BLKW. 1,R.GID,GBL  ;Region ID
     .BLKW. 1,R.GSIZ,GBL ;Size of region (32W blocks)
     .BLKW. 2,R.GNAM,GBL ;Name of region (RAD50)
     .BLKW. 2,R.GPAR,GBL ;Region's main partition name (RAD50)
     .BLKW. 1,R.GSTS,GBL ;Region status word
     .BLKW. 1,R.GPRO,GBL ;Protection code of region
     .BLKW. 0,R.GLGH,GBL ;Length of region definition block
```

8.4.12.1 Region Status Word Symbols

```
              .IF IDN <DEF$G>,<GBL>
     .GLOBL RS.CRR,RS.UNM,RS.MDL,RS.NDL,RS.ATT,RS.NEX
     .GLOBL RS.DEL,RS.EXT,RS.WRT,RS.RED
              .ENDC
```

```
RS.CRR=^O<100000>        ;Region was successfully created
RS.UNM=^O<40000>         ;One or more windows were unmapped on a
                         ; detach
RS.MDL=^O<200>           ;Mark region for delete on last detach
RS.NDL=^O<100>           ;Created region is not to be marked for
                         ; detach
RS.ATT=^O<40>            ;Attach to created region
RS.NEX=^O<20>            ;Created region is not extendable
RS.DEL=^O<10>            ;Delete access desired on attach
RS.EXT=^O<4>             ;Extend access desired on attach
RS.WRT=^O<2>             ;Write access desired on attach
RS.RED=^O<1>             ;Read access desired on attach
```

8.4.13 Status Control Block (SCB)

Defined by: .MACRO SCBDF$,L,B,SYSDEF

The SCB defines the status of a device controller. The SCB is pointed to by Unit Control Blocks (UCBs). One SCB exists for each device controller in the system. This is true even if the controller handles more than one device-unit (as in the case of the RK11 Controller). However, line multiplexers such as the DH11 and DJ11 are considered to have one controller for each line because all lines may transfer in parallel. As an example, where a group of terminals may be connected to two controllers, a DL11-A and a DH11, each terminal interfaced via the DL11-A would have a SCB because each DL11-A is an independent interface unit. The terminals interfaced via the DH11 would also each have an SCB because the DH11 is a single controller but multiplexes many units in parallel.

Most of the information in the SCB is dynamic, and is used by both the Executive and the driver.

```
     .ASECT
.=177772
S.RCNT:'L'     .BLKB 1    ;Number of registers to copy on error
S.ROFF:'L'     .BLKB 1    ;Offset to first device register
S.BMSV:'L'     .BLKW 1    ;Saved I/O active bitmap and pointer to
                            the EMB
S.BMSK:'L'     .BLKW 1    ;Device I/O active bit mask
S.LHD:'L'      .BLKW 2    ;Controller I/O queue listhead.  The first
                            word points to the first I/O packet in the
                            queue and the second word points to the
                            last I/O packet in the queue.  If the queue
                            is empty, the first word is 0 and the
                            second word points to the first word.
S.PRI:'L'      .BLKB 1    ;Priority at which the device interrupts.
S.VCT:'L'      .BLKB 1    ;Interrupt vector address divided by 4.
                            For loadable drivers, the MCR/VMR LOA[D]
                            function uses this field and the existence
                            of driver symbol(s) $xxINT, $xxINP, and
                            $xxOUT to initialize the device interrupt
                            vector.
S.CM:'L'      .BLKB 1    ;Current timeout count.  RSX-11M supports
                            device timeout, which enables a driver to
                            limit the time that elapses between the
                            issuing of an I/O operation and its
                            termination.  The current timeout count (in
                            seconds) is initialized by moving S.ITM
                            (initial timeout count) into S.CTM.  The
                            Executive clock routines (in TDSCH) examine
```

```
                                 the time, decrement it and, if it reaches 0,
                                 call the driver at its device
                                 timeout entry point.
                                   The internal clock count is kept in
                                 1-second increments.  Thus, a time count of
                                 1 is not precise because the internal
                                 clocking mechanism is operating asynchro-
                                 nously with driver execution. The minimum
                                 meaningful clock interval is 2 if you
                                 intend to treat timeout as a consistently
                                 detectable error condition.  Note, if the
                                 count is 0, that no timeout occurs; a 0
                                 value is, in fact, an indication that
                                 timeout is not operative.  The maximum
                                 count is 255.  You are responsible for
                                 setting this field.  Resetting occurs at
                                 actual timeout or within $FORK.
S.ITM:'L'      .BLKB 1          ;Initial timeout value
S.CON:'L'      .BLKB 1          ;Controller index. This is the controller
                                 number multiplied by 2.  Drivers that are
                                 written to support more than one controller
                                 use this byte.  S.CON may be used by the
                                 driver to index into a controller table
                                 that the driver creates and maintains
                                 internally.  Indexing the controller table
                                 enables the driver to service the correct
                                 controller when a device causes an
                                 interrupt.
S.STS:'L'      .BLKB 1          ;Controller status (0=idle, 1=busy).
                                 This is the interlock for marking a driver
                                 as busy for a specific controller.  It is
                                 tested and set by $GTPKT and reset by
                                 $IODON.
S.CSR:'L'      .BLKW 1          ;Address of the control status register
                                 for the device controller.  The driver uses
                                 S.CSR to initiate I/O operations and to
                                 access, by indexing, other registers re-
                                 lated to the device that are located in
                                 the I/O page.  This address need not be a
                                 CSR.  It may only be a member of the
                                 device's register set.  It is accessed at
                                 system bootstrap time to determine if the
                                 interface exists on the system hosting the
                                 Executive.  The Executive uses S.CSR to set
                                 the offline bit at bootstrap so that system
                                 software can be interchanged between systems
                                 without an intervening system generation.
                                 Otherwise, it is only accessed by the driver
                                 itself.
S.PKT:'L'      .BLKW 1          ;Address of current I/O packet established by
                                 $GTPKT.  This field is used to retrieve the
                                 I/O packet address upon the completion of
                                 an I/O request.  S.PKT is not zeroed after
                                 the packet is completed.
S.FRK:'L'      .BLKW 1          ;Fork block link word
               .BLKW 1          ;Fork-PC
               .BLKW 1          ;Fork-R5
               .BLKW 1          ;Fork-R4
                                 The four words starting at S.FRK are used
                                 for fork block storage when the driver es-
                                 tablishes itself as a fork process.  Fork
                                 block storage preserves the state of the
                                 driver, which is restored when the driver
                                 regains control at fork level.  This area
                                 is used when the driver calls $FORK.
```

```
                          The fork block is 5 words long in-
                          stead of 4 if two conditions are met:
                           1.  Loadable drivers have been selected as
                               a SYSGEN option
                           2.  The system is a mapped system
                          If these conditions are met, and the fork
                          block is 5 words long, you must not use
                          the fork block for any other purpose.  In
                          other words, your driver may not share the
                          space reserved for the fork block.  If it is
                          shared, the loadable driver's relocation
                          base will be destroyed.  In addition, the
                          5-word fork block should always be part of
                          the SCB if the two conditions listed above
                          are met.
              .IF NB SYSDEF
              .IF DF L$$DRV & M$$MGE
              .BLKW 1   ;Fork driver relocation base
              .ENDC

S.CCB:'L'               ;Mixed MASSBUS channel control block
S.MPR:'L'     .BLKW 6   ;11/70 extended memory UNIBUS device
                          C-block.  Drivers use the 6 words starting
                          at S.MPR for non-processor request (NPR)
                          devices attached to a PDP-11/70 with 22-bit
                          addressing.
              .IFF
     .PSECT
```

8.4.13.1 Status Control Block Priority Byte Condition Code Status Bit Definition

```
SP.EIP='B'1             ;Error in progress (1=yes)
SP.ENB='B'2             ;Error logging enabled (0=yes)
SP.LOG='B'4             ;Error logging available (1=yes)
SPARE=10                ;Spare bit
```

8.4.13.2 Mapping Assignment Block (for UNIBUS Mapping Register Assignment)

```
     .ASECT
.=0
M.LNK:'L'     .BLKW 1   ;Link word
M.UMRA:'L'    .BLKW 1   ;Address of first assigned UMR
M.UMRN:'L'    .BLKW 1   ;Number of UMR's assigned * 4
M.UMVL:'L'    .BLKW 1   ;Low 16 bits mapped by 1st assigned UMR
M.UMVH:'L'    .BLKB 1   ;High 2 bits mapped in bits 4 and 5
M.BFVH:'L'    .BLKB 1   ;High 6 bits of physical buffer address
M.BFVL:'L'    .BLKW 1   ;Low 16 bits of physical buffer address
M.LGTH='B'.             ;Length of mapping assignment block
              .ENDC
     .PSECT
```

8.4.14 Snap Block

```
Defined by: .MACRO SNPDF$ GBL
           .IF IDN <GBL>,<DEF$G>
     .GLOBL SB.CTL,SB.DEV,SB.UNT,SB.EFN,SB.ID,SB.LM1,SB.PMD
     .GLOBL SC.HDR,SC.LUN,SC.OVL,SC.STK,SC.WRD,SC.BYT
           .ENDC
```

```
SB.CTL = 0              ;Control word
SB.DEV = 2              ;Device name
SB.UNT = 4              ;Device unit number
SB.EFN = 6              ;Event flag number
SB.ID  = ^O<10>         ;Snapshot identification word
SB.LM1 = ^O<12>         ;First word of address limits
SB.PMD = ^O<32>         ;.RAD50 /PMD.../
SC.HDR = 1              ;Dump task header (including registers)
SC.LUN = 2              ;Dump assigned LUN information
SC.OVL = 4              ;Dump overlay structure information
SC.STK = ^O<10>         ;Dump task stack
SC.WRD = ^O<20>         ;Send output in word/RAD50 format
SC.BYT = ^O<40>         ;Send output in byte/ASCII format
```

8.4.15 Task Abort Codes

Defined by: .MACRO ABODF$,L,B

NOTE: S.COAD-S.CFLT are also SST vector offsets

```
S.COAD='B'0.            ;Odd address and traps to 4
S.CSGF='B'2.            ;Segment fault
S.CBPT='B'4.            ;Break point or trace trap
S.CIOT='B'6.            ;IOT instruction
S.CILI='B'8.            ;Illegal or reserved instruction
S.CEMT='B'10.           ;Non RSX EMT instruction
S.CTRP='B'12.           ;TRAP instruction
S.CFLT='B'14.           ;11/40 floating point exception
S.CSST='B'16.           ;SST abort - bad stack
S.CAST='B'18.           ;AST abort - bad stack
S.CABO='B'20.           ;ABORT via directive
S.CLRF='B'22.           ;Task load request failure
S.CCRF='B'24.           ;Task checkpoint read failure
S.IOMG='B'26.           ;Task exit with outstanding I/O
S.PRTY='B'28.           ;Task memory parity error
              .ENDM
```

8.4.16 Task Control Block (TCB) And Status Definitions

Defined by: .MACRO TCBDF$,L,B,SYSDEF

```
T.LNK:'L'     .BLKW 1   ;Utility link word
T.PRI:'L'     .BLKB 1   ;Task priority
T.IOC:'L'     .BLKB 1   ;I/O pending count
T.CPCB:'L'    .BLKW 1   ;Pointer to checkpoint PCB
T.NAM:'L'     .BLKW 2   ;Task name in RAD50
T.RCVL:'L'    .BLKW 2   ;Receive queue listhead
T.ASTL:'L'    .BLKW 2   ;AST queue listhead
T.EFLG:'L'    .BLKW 2   ;Task local event flags 1-32
T.UCB:'L'     .BLKW 1   ;UCB address for pseudo device 'TI'
T.TCBL:'L'    .BLKW 1   ;Task list thread word
T.STAT:'L'    .BLKW 1   ;First status word (blocking bits)
T.ST2:'L'     .BLKW 1   ;Second status word (state bits)
T.ST3:'L'     .BLKW 1   ;Third status word (attribute bits)
T.DPRI:'L'    .BLKB 1   ;Task's default priority
T.LBN:'L'     .BLKB 3   ;LBN of task load image
T.LDV:'L'     .BLKW 1   ;UCB address of load device
T.PCB:'L'     .BLKW 1   ;PCB address of task partition
```

```
T.MXSZ:'L'    .BLKW 1   ;Maximum size of task image (mapped only)
T.ACTL:'L'    .BLKW 1   ;Address of next task in active list

;START OF PLAS AREA

T.ATT:'L'     .BLKW 2   ;Attachment descriptor listhead
T.OFF:'L'     .BLKW 1   ;Offset to task image in partition
              .BLKB 1   ;Reserved
T.SRCT:'L'    .BLKB 1   ;SREF with EFN count in all receive
                        ; queues
T.RRFL:'L'    .BLKW 2   ;Receive by reference listhead
              .IF NB SYSDEF
              .IF NDF P$$LAS
T.LGTH='B'T.ATT
              .IFF
T.LGTH='B'.             ;Length of task control block
              .ENDC
T.EXT='B'0              ;Length of TCB extension
```

8.4.16.1 **Task Status Definitions**

First Status Word (Blocking Bits)

```
TS.EXE='B'100000        ;Task not in execution (1=yes)
TS.RDN='B'40000         ;I/O rundown in progress (1=yes)
TS.MSG='B'20000         ;ABORT message being displayed (1=yes)
TS.NRP='B'10000         ;Task mapped to nonresident partition
                          (1=yes)
TS.RUN='B'4000          ;Task is running on another processor
                          (1=yes)
TS.OUT='B'400           ;Task is out of memory (1=yes)
TS.CKP='B'200           ;Task is being checkpointed (1=yes)
TS.CKR='B'100           ;Task checkpoint requested (1=yes)
```

Task Blocking Status Mask

```
TS.BLK='B'TS.CKP!TS.CKR!TS.EXE!TS.MSG!TS.NRP!TS.OUT!TS.RDN
```

Second Status Word (State Bits)

```
T2.AST='B'100000        ;AST in progress (1=yes)
T2.DST='B'40000         ;AST recognition disabled (1=yes)
T2.CHK='B'20000         ;Task not checkpointable (1=yes)
T2.CKD='B'10000         ;Checkpointing disabled (1=yes)
T2.BFX='B'4000          ;Task being fixed in memory (1=yes)
T2.FXD='B'2000          ;Task fixed in memory (1=yes)
T2.TIO='B'1000          ;Task is engaged in terminal I/O
T2.CAF='B'400           ;DYN checkpoint space allocation failure
T2.HLT='B'200           ;Task is being halted (1=yes)
T2.ABO='B'100           ;Task marked for ABORT (1=yes)
T2.STP='B'40            ;Task stopped (1=yes)
T2.STP='B'20            ;Task stopped (1=yes)
T2.SPN='B'10            ;Saved TS.SPN on AST in progress
T2.SPN='B'4             ;Task suspended (1=yes)
T2.WFR='B'2             ;Saved TS.WFR on AST in progress
T2.WFR='B'1             ;Task in Wait For state (1=yes)
```

Third Status Word (Attribute Bits)

```
T3.ACP='B'100000        ;Ancillary control processor (1=yes)
T3.PMD='B'40000         ;Dump task on synchronous abort (0=yes)
T3.REM='B'20000         ;Remove task on exit (1=yes)
T3.PRV='B'10000         ;Task is privileged (1=yes)
T3.MCR='B'4000          ;Task requested as external MCR function
                          (1=yes)
T3.SLV='B'2000          ;Task is a slave task (1=yes)
T3.CLI='B'1000          ;Task is a command line interpreter
                          (1=yes)
T3.RST='B'400           ;Task is restricted (1=yes)
T3.NSD='B'200           ;Task does not allow send data
T3.CAL='B'100           ;Task has checkpoint space in task image
T3.ROV='B'40            ;Task has resident overlays
T3.NET='B'20            ;Network protocol level
```

8.4.17 Task Header

Defined by: .MACRO HDRDF$,L,B

```
     .ASECT
.=0
H.CSP:'L'      .BLKW 1   ;Current stack pointer
H.HDLN:'L'     .BLKW 1   ;Header length in bytes
H.EFLM:'L'     .BLKW 2   ;Event flag mask word and address
H.CUIC:'L'     .BLKW 1   ;Current task UIC
H.DUIC:'L'     .BLKW 1   ;Default task UIC
H.IPS:'L'      .BLKW 1   ;Initial processor status word (PS)
H.IPC:'L'      .BLKW 1   ;Initial program counter (PC)
H.ISP:'L'      .BLKW 1   ;Initial stack pointer (SP)
H.ODVA:'L'     .BLKW 1   ;ODT SST vector address
H.ODVL:'L'     .BLKW 1   ;ODT SST vector length
H.TKVA:'L'     .BLKW 1   ;Task SST vector address
H.TKVL:'L'     .BLKW 1   ;Task SST vector length
H.PFVA:'L'     .BLKW 1   ;Power fail AST control block address
H.FPVA:'L'     .BLKW 1   ;Floating point AST control block address
H.RCVA:'L'     .BLKW 1   ;Receive AST control block address
H.EFSV:'L'     .BLKW 1   ;Event flag address save address
H.FPSA:'L'     .BLKW 1   ;Pointer to floating point/EAE save area
H.WND:'L'      .BLKW 1   ;Pointer to number of window blocks
H.DSW:'L'      .BLKW 1   ;Task directive status word
H.FCS:'L'      .BLKW 1   ;FCS impure pointer
H.FORT:'L'     .BLKW 1   ;FORTRAN impure pointer
H.OVLY:'L'     .BLKW 1   ;Overlay impure pointer
H.VEXT:'L'     .BLKW 1   ;Work area extension vector pointer
H.SPRI:'L'     .BLKB 1   ;Priority difference for swapping
H.NML:'L'      .BLKB 1   ;Network mailbox LUN
H.RRVA:'L'     .BLKW 1   ;Receive by reference AST control block
                         ; address
               .BLKW 3   ;Reserved words
H.GARD:'L'     .BLKW 1   ;Pointer to header guard word
H.NLUN:'L'     .BLKW 1   ;Number of LUNs
H.LUN:'L'      .BLKW 2   ;Start of logical unit table
```

8.4.17.1 Window Block Offsets

```
.=0
W.BPCB:'L'     .BLKW 1   ;Partition control block address
W.BLVR:'L'     .BLKW 1   ;Low virtual address limit
```

```
W.BHVR:'L'     .BLKW 1    ;High virtual address limit
W.BATT:'L'     .BLKW 1    ;Address of attachment descriptor
W.BSIZ:'L'     .BLKW 1    ;Size of window in 32W blocks
W.BOFF:'L'     .BLKW 1    ;Physical memory offset in 32W blocks
W.BFPD:'L'     .BLKB 1    ;First PDR address
W.BNPD:'L'     .BLKB 1    ;Number of PDRs to map
W.BLPD:'L'     .BLKW 1    ;Contents of last PDR
W.BLGH:'L'                ;Length of window descriptor
```

8.4.18 Task Image File Label Block

8.4.18.1 Resident Library Descriptor Offsets

Defined by: .MACRO LBLDF$,L,B

```
     .ASECT
.=0
R$LNAM:'L'     .BLKW 2    ;RADIX-50 library name
R$LSA:'L'      .BLKW 1    ;Library starting virtual address
R$LHGV:'L'     .BLKW 1    ;Library address window 0 bound
R$LMXV:'L'     .BLKW 1    ;Library high virtual address limit
R$LLDZ:'L'     .BLKW 1    ;Library load size (32W blocks)
R$LMXZ:'L'     .BLKW 1    ;Library max. size (32W blocks)
R$LOFF:'L'     .BLKW 1    ;Library offset into partition (32W
                          ; blocks)
R$LWND:'L'     .BLKW 1    ;Number of library address windows
R$LSEG:'L'     .BLKW 1    ;Size of library segment descriptors
R$LFLG:'L'     .BLKW 1    ;Library flags word
R$LDAT:'L'     .BLKW 3    ;Library creation date (yr., mo., day)
R$LSIZ:'L'     .BLKW 0    ;Length of library descriptor
```

8.4.18.2 Library List Entry Flags

```
LD$ACC='B'100000          ;Access intent (1=RW, 0=RO)
LD$RSV='B'040000          ;APR reservation flag (1=APR reserved)
LD$REL='B'000004          ;PIC flag (1=position independant)
```

8.4.18.3 Label Block Offsets

```
.=0
L$BTSK:'L'     .BLKW 2    ;RADIX 50 task name
```

NOTE

Label block parameters between this offset and the start of the task library descriptors must be identical in format and content to a resident library descriptor entry.

```
L$BPAR:'L'     .BLKW 2      ;RADIX 50 partition name
L$BSA:'L'      .BLKW 1      ;Starting address of task
L$BHGV:'L'     .BLKW 1    ;Window 0 virtual address limit
L$BMXV:'L'     .BLKW 1    ;Task high virtual address limit
```

```
L$BLDZ:'L'     .BLKW 1    ;Task load size (32W blocks)
L$BMXZ:'L'     .BLKW 1    ;Task max. size (32W blocks)
L$BOFF:'L'     .BLKW 1    ;Task offset into partition (32W blocks)
L$BWND:'L'     .BLKW 1    ;Number of task windows (less libraries)
L$BSEG:'L'     .BLKW 1    ;Size of task segment descriptors (bytes)
L$BFLG:'L'     .BLKW 1    ;Task flags word
L$BDAT:'L'     .BLKW 3    ;Task creation date (yr., mo., day)
L$BLIB:'L'     .BLKW <7.*<R$LSIZ/2>>+1 ;Resident library entries
L$BPRI:'L'     .BLKW 1    ;Task priority
L$BXFR:'L'     .BLKW 1    ;Task transfer address
L$BEXT:'L'     .BLKW 1    ;Task extend size (32W blocks)
L$BSGL:'L'     .BLKW 1    ;Relative block number of segment length
                          ; list
L$BHRB:'L'     .BLKW 1    ;Relative block number of task image
                          ; header
L$BBLK:'L'     .BLKW 1    ;Number of blocks in label
L$BLUN:'L'     .BLKW 1    ;Number of logical units
               .BLKW <512.-.>/2
L$BASG:'L'     .BLKW 0    ;Start of device assignment tables
TS$PIC='B'100000          ;Task is PIC (1=yes)
TS$NHD='B'040000          ;No header in task image (1=yes)
TS$ACP='B'020000          ;Task is ancillary control processor (1=yes)
TS$PMD='B'010000          ;Generate post-mortem dump (1=yes)
TS$SLV='B'004000          ;Task is slaveable (1=yes)
TS$NSD='B'002000          ;No send to task is permitted (1=yes)
TS$NET='B'001000          ;Task uses new network protocol (1=yes)
TS$PRV='B'000400          ;Task is privileged (1=yes)
TS$CMP='B'000200          ;Task built in compatibility mode (1=yes)
TS$CHK='B'000100          ;Task is checkpointable (0=yes)
TS$RES='B'000040          ;Task has resident overlays (1=yes)
     .PSECT
```

8.4.19 Task Termination Notification Message Codes

Defined by: .MACRO ABODF$,L,B

```
T.NDNR='B'0               ;Device not ready
T.NDSE='B'2               ;Device select error
T.NCWF='B'4               ;Checkpoint write failure
T.NCRE='B'6               ;Card reader hardware error
T.NDMO='B'8.              ;Dismount complete
T.NLDN='B'12.             ;Link down (networks)
T.NLUP='B'14.             ;Link up (networks)
            .ENDM
```

8.4.20 Unit Control Block (UCB)

Defined by: .MACRO UCBDF$,L,B

One UCB exists for each device-unit attached to a system. In other words, one UCB exists for each device-unit of each DCB. The UCB defines the status of an individual device-unit, and is the control block that is pointed to by the first word of an assigned LUN. The UCBs associated with a particular DCB are contiguous in memory, have the same length, and are pointed to by the DCB. UCBs associated with different DCBs may have different lengths but are of the same length for a specific DCB.

Much of the information in the UCB is static, though a few dynamic parameters exist. From the UCB, however, it is possible to access most of the other structures in the I/O data base. In this sense, the

UCB gives access to a large amount of dynamic data. For example, the redirect pointer, which reflects the results of an MCR Redirect command, is updated dynamically.

As with the DCB, most of the UCB is established in the assembly source; however, its contents are used and modified by both the Executive and the driver, though modification of a given datum is usually done by either the Executive or the driver, but not both. Because the UCB is the key control in the I/O data structures, access to other I/O control blocks usually occurs via links implanted in the UCB.

```
     .ASECT
.=177774
U.LUIC:'L'     .BLKW 1    ;LOGIN UIC - for terminal UCBs on multi-user
                            systems only
U.OWN:'L'      .BLKW 1    ;The UCB address of the owning terminal for
                            allocated devices - multi-user systems
                            only
U.DCB:'L'      .BLKW 1    ;Back pointer to corresponding DCB
                            Access to other control blocks in the I/O
                            data structure usually occurs via the UCB.
U.RED:'L'      .BLKW 1    ;Pointer to redirect unit UCB. initially
                            points to U.DCB.  This field is changed
                            as the result of the Redirect command.
                            After the command is issued, this field
                            points to the UCB to which this device-unit
                            has been redirected.  The redirect
                            chain ends when this field points to U.DCB
                            field in the UCB in which it resides.
U.CTL:'L'      .BLKB 1    ;Control processing flags (set at assembly
                            time).  U.CTL and the function mask words
                            in the DCB drive QIO directive processing.
                            Any inaccuracy in the bit setting of U.CTL
                            produces erroneous I/O processing. See the
                            RSX-11M Guide to Writing an I/O Driver for
                            more details.
U.STS:'L'      .BLKB 1    ;Device independent unit status
U.UNIT:'L'     .BLKB 1    ;Physical unit number of device.  If the
                            controller for the device supports only one
                            unit, the unit number is always 0.
U.ST2:'L'      .BLKB 1    ;Unit status extension

U.CW1:'L'      .BLKW 1    ;First device characteristics word
                             This is the first word in a cluster of
                            device characteristics information.  U.CW1
                            and U.CW4 are device independent.  U.CW2
                            and U.CW3 are device dependent.  The four
                            characteristic words are retrieved from the
                            UCB and placed in the requestor's buffer
                            upon issuance of a GLUN$ Executive direc-
                            tive.  It is the responsibility of the
                            driver writer to supply the contents of
                            these four words in the assembly source of
                            the driver's data structure.  See the
                            RSX-11M Guide to Writing an I/O Driver.
U.CW2:'L'      .BLKW 1    ;Second device characteristics word.  This
                            word is specific to a given device driver
                            and, with an exception, is available for
                            working storage or constants.  The ex-
                            ception is for block-structured devices.
                            In this case, U.CW2 and U.CW3 may not be
                            used for working storage.  In drivers for
                            block-structured devices (disks and
                                DECtape), these two words must be
                                initialized to a double-precision number
                                giving the total number of blocks on the
                                device.  Place the high-order bits in the
                                low-order byte of U.CW2 and the low-order
                                bits in U.CW3.
U.CW3:'L'     .BLKW 1         ;Third device characteristics word
U.CW4:'L'     .BLKW 1         ;Fourth device characteristics word
U.SCB:'L'     .BLKW 1         ;Pointer to SCB  for this UCB. In general, R4
                                contains the value in this word upon entry
                                to the driver via the driver dispatch table
                                because service routines frequently refer-
                                ence the SCB.
U.ATT:'L'     .BLKW 1         ;Address of the TCB of the task attached to
                                to the unit.
U.BUF:'L'     .BLKW 1         ;Relocation bias of current I/O request.
              .BLKW 1         ;Buffer address of current I/O request.
                                U.BUF labels two consecutive words that
                                serve as a communication region between
                                $GTPKT and the driver.  If a non-transfer
                                function is indicated (in D.MSK), U.BUF,
                                U.BUF+2, and U.CNT receive the first three
                                parameter words from the I/O packet.
                                  For transfer operations, the format of
                                these two words depends upon the setting of
                                UC.NPR in U.CTL.  The driver does not format
                                the words; all formatting is completed be-
                                fore the driver receives control.  For un-
                                mapped systems, the first word is 0 and
                                the second word is the physical address of
                                the buffer,  For mapped systems, the UC.NPR
                                bit determines the format.  UC.NPR is set
                                for an NPR device and reset for a pro-
                                gram-transfer device.
                                  For more information, see the RSX-11M
                                Guide to Writing an I/O Driver.
U.CNT:'L'     .BLKW 1         ;Byte count of current I/O request
                                Contains the byte count of the buffer
                                described by U.BUF.  The driver uses this
                                field to construct the device address.
                                  U.BUF and U.CNT keep track of the current
                                data item in the buffer for the current
                                transfer (except for NPR transfers).  Be-
                                cause this field is being altered dyna-
                                mically, the I/O packet may be needed to
                                reissue an I/O operation; for instance,
                                after a powerfail or error retry.
U.VCB='B'U.CNT+4              ;Address of volume control block
U.CBF='B'U.CNT+2              ;Control buffer relocation and address
U.UIC='B'U.CNT+<9.*2>         ;Terminal UIC (terminals only)
      .PSECT
```

8.4.20.1 Device Table Status Definitions

Device Characteristics Word 1 (U.CW1) Device Type Definition Bits

```
DV.REC='B'1                   ;Record oriented device (1=yes)
DV.CCL='B'2                   ;Carriage control device (1=yes)
DV.TTY='B'4                   ;Terminal device (1=yes)
DV.DIR='B'10                  ;File structured device (1=yes)
DV.SDI='B'20                  ;Single directory device (1=yes)
DV.SQD='B'40                  ;Sequential device (1=yes)
DV.MXD='B'100           ;MASSBUS device (1=yes)
DV.UMD='B'200           ;User mode diagnostics supported (1=yes)
DV.SWL='B'1000          ;Unit software write locked (1=yes)
DV.ISP='B'2000          ;Input spooled device (1=yes)
DV.OSP='B'4000          ;Send output to spooled device (1=yes)
DV.PSE='B'10000         ;Pseudo device (1=yes)
DV.COM='B'20000         ;Device is mountable as COM channel
                        ; (1=yes)
DV.F11='B'40000         ;Device is mountable as F11 device
                        ; (1=yes)
DV.MNT='B'100000        ;Device is mountable (1=yes)
```

Terminal Dependent Characteristics Word 2 (U.CW2) Bit Definitions

```
U2.DH1='B'100000        ;Unit is a multiplexer (1=yes)
U2.DJ1='B'40000         ;Unit is a DJ11 (1=yes)
U2.RMT='B'20000         ;Unit is remote (1=yes)
U2.L8S='B'10000         ;Unit is LA180s (1=yes)
U2.NEC='B'4000          ;Do not echo solicited input (1=yes)
U2.CRT='B'2000          ;Unit is a CRT (1=yes)
U2.ESC='B'1000          ;Unit generates escape sequences (1=yes)
U2.LOG='B'400           ;User logged on terminal (0=yes)
U2.SLV='B'200           ;Unit is a slave terminal (1=yes)
U2.DZ1='B'100           ;Unit is a DZ11 (1=yes)
U2.HLD='B'40            ;Terminal is in hold screen mode (1=yes)
U2.AT.='B'20            ;MCR command AT. being processed (1=yes)
U2.PRV='B'10            ;Unit is a privileged terminal (1=yes)
U2.L3S='B'4             ;Unit is a LA30S terminal (1=yes)
U2.VT5='B'2             ;Unit is a VT05B terminal (1=yes)
U2.LWC='B'1             ;Lower case to upper case conversion
                        ; (1=yes)
```

RH11-RS03/RS04 Characteristics Word 2 (U.CW2) Bit Definitions

```
U2.R04='B'100000        ;Unit is a RS04 (1=yes)
```

RH11-TU16 Characteristics Word 2 (U.CW2) Bit Definitions

```
U2.7CH='B'10000         ;Unit is a 7 channel drive (1=yes)
```

Unit Control Processing Flag Definitions

```
UC.ALG='B'200           ;Byte alignment of data buffers is allowed
                          (0=yes)
                          Word alignemnt is allowed (1=yes)
UC.NPR='B'100           ;Device is an NPR device (1=yes)
                          This word determines the format of the
                          2-word address in U.BUF.
UC.QUE='B'40            ;Call driver before queuing (1=yes)
                          If set, the QIO directive processor calls
                          the driver prior to queing the I/O packet.
                          The disposition of the I/O packet is the
                          driver's responsibility.  Typically, an I/O
                          packet is queued prior to a call to the
                          driver, which later retrieves it by a call
                          to $GTPKT.
UC.PWF='B'20            ;Call driver at powerfail always (1=yes)
                          If set, the driver is always called when
                          power is restored after a power failure
                          occurs.  Typically, the driver is called on
                          power restoration only when an I/O opera-
                          tion is in progress.
```

```
UC.ATT='B'10            ;Call driver on ATTACH/DETACH (1=yes)
                          If this bit is set, the driver is called
                          when $GTPKT processes an Attach/Detach I/O
                          function.  Typically, the driver does not
                          get control for Attach/Detach requests and
                          the Executive performs the entire function
                          without any assistance from the driver.
UC.KIL='B'4             ;Call driver at I/O kill always (1=yes)
                          If set, the driver is called on a Cancel
                          I/O request even if the specified unit is
                          not busy.  Typically, the driver is called
                          on a Cancel I/O only if an I/O operation
                          is in progress.
UC.LGH='B'3             ;Transfer length mask bits
                          These two bits are used to check
                          whether the byte count specified
                          in an I/O request is a legal buffer
                          modulus. See Guide to Writing an
                          I/O Driver manual.
```

Unit Status (U.STS) Bit Defintions

This byte contains device-independent status information. US.MDM, US.MNT, and US.FOR apply only to mountable devices.

```
US.BSY='B'200           ;Unit is busy (1=yes)
US.MNT='B'100           ;Unit is mounted (0=yes)
US.FOR='B'40            ;Unit is mounted as foreign volume (1=yes)
US.MDM='B'20            ;Unit is marked for dismount (1=yes)
```

Card Reader Dependent Unit Status Bit Definitions

```
US.ABO='B'1             ;Unit is marked for abort if not ready
                        ; (1=yes)
US.MDE='B'2             ;Unit is in 029 translation mode (1=yes)
```

FILES-11 Dependent Unit Status Bits

```
US.WCK='B'10            ;Write check enabled (1=yes)
US.SPU='B'2             ;Unit is spinning up (1=yes)
```

Terminal Dependent Unit Status Bit Definitions

```
US.DSB='B'10            ;Unit is disabled (1=yes)
US.CRW='B'4             ;Unit is waiting for carrier (1=yes)
US.ECH='B'2             ;Unit has echo in progress (1=yes)
US.OUT='B'1             ;Unit is expecting output interrupt
                        ; (1=yes)
```

LPS11 Dependent Unit Status Bit Definitions

```
US.FRK='B'2             ;Fork in progress (1=yes)
US.SHR='B'1             ;Shareable function in progress (0=yes)
```

ANSI Magtape Dependent Unit Status Bits

```
US.LAB='B'4             ;Unit has labeled tape on it (1=yes)
```

Unit Status Extension (U.ST2) Bit Definitions

```
US.OFL='B'1             ;Unit offline (1=yes)
US.RED='B'2             ;Unit redirectable (0=yes)
US.PUB='B'4             ;Unit is public device (1=yes)
US.UMD='B'10            ;Unit attached for diagnostics (1=yes)
```

8.4.21 Volume Control Block (VCB)

```
     .ASECT
.=0
V.TRCT:       .BLKW 1   ;Transaction count
V.IFWI:       .BLKW 1   ;Index file window
V.FCB:        .BLKW 2   ;File Control Block listhead
V.IBLB:       .BLKB 1   ;Index bit map 1st LBN high byte
V.IBSZ:       .BLKB 1   ;Index bit map size in blocks
              .BLKW 1   ;Index bit map 1st LBN low bits
V.FMAX:       .BLKW 1   ;Max no. of files on volume
V.WISZ:       .BLKB 1   ;Dflt size of window in no. of rtrv ptrs
                        ;Value is < 128.
V.SBCL:       .BLKB 1   ;Storage bit map cluster factor
V.SBSZ:       .BLKW 1   ;Storage bit map size in blocks
V.SBLB:       .BLKB 1   ;Storage bit map 1st LBN high byte
V.FIEX:       .BLKB 1   ;Default file extend size
              .BLKW 1   ;Storage bit map 1st LBN low bits
V.VOWN:       .BLKW 1   ;Volume owner's UIC
V.VPRO:       .BLKW 1   ;Volume protection
V.VCHA:       .BLKW 1   ;Volume characteristics
V.FPRO:       .BLKW 1   ;Volume default file protection
V.VFSQ:       .BLKW 1   ;Volume file sequence number
V.FRBK:       .BLKB 1   ;Number of free blocks on volume high
                          byte
V.LRUC:       .BLKB 1   ;Count of available LRU slots in FCB list
              .BLKW 1   ;Number of free blocks on volume low bits
V.STAT:       .BLKB 1   ;Volume status byte, containing the
                          following:
VC.IFW= 1                 Index file is write accessed
VC.BMW= 2                 Storage bit map file is write accessed
V.FFNU:       .BLKB 1   ;First free index file bit map block
V.LGTH:                 ;Size in bytes of VCB
```

8.4.22 Window Definition Block (WDB)

Defined by: .MACRO WDGDF$ GBL

8.4.22.1 Window Definition Block Offsets

```
  .MCALL  .BLKW.,.BLKB.,.BLK.
  .BLK.
  .BLKB.  1,W.NID,GBL       ;Window ID
  .BLKB.  1,W.NAPR,GBL      ;Base APR
  .BLKW.  1,W.NBAS,GBL      ;Virtual base address (bytes)
  .BLKW.  1,W.NSIZ,GBL      ;Window size (32W blocks)
  .BLKW.  1,W.NRID,GBL      ;Region ID
  .BLKW.  1,W.NOFF,GBL      ;Offset in partition (32W blocks)
  .BLKW.  1,W.NLEN,GBL      ;Length to map (32W blocks)
  .BLKW.  1,W.NSTS,GBL      ;Window status word
  .BLKW.  1,W.NSRB,GBL      ;Send/receive buffer virtual address (bytes)
  .BLKW.  0,W.NLGH,GBL      ;Length of window definition block
```

8.4.22.2 Window Status Word Symbols

```
  .IF IDN <DEF$G>,<GBL>
  .GLOBL  WS.CRW,WS.UNM,WS.ELW,WS.RRF,WS.64B
  .GLOBL  WS.MAP,WS.RCX,WS.DEL,WS.EXT,WS.WRT,WS.RED
  .ENDC

WS.CRW=^O<100000>          ;Address window was successfully created
WS.UNM=^O<40000>           ;One or more windows were unmapped in
                           ;Create address window or map.
WS.ELW=^O<20000>           ;One or more windows were eliminated in
                           ;Create address window
WS.RRF=^O<10000>           ;Reference was successfully received
WS.64B=^o<400>             ;64 byte alignment allowed
WS.MAP=^O<200>             ;Map after create window or receive
                           by reference
WS.RCX=^O<100>             ;Exit if no references to receive
WS.DEL=^O<10>              ;Send with delete access
WS.EXT=^O<4>               ;Send with extend access
WS.WRT=^O<2>               ;Send with write access or map with
                           ; write access
WS.RED=^O<1>               ;Send with read access
```

CHAPTER 9

CROSS-REFERENCES

9.1 EXECUTIVE MODULE TO ROUTINE CROSS-REFERENCE

This cross-reference contains a listing of the executive modules (driver tables not included) and the routines that they contain. The routines are in alphabetical order as are the modules. A dollar sign ($) preceeds the label of global routines. All named labels are in this cross-reference but some are the labels of data areas or fields.

Large and important local routines are in this cross-reference. A dollar sign ($) does not preceed these routines because they are not global.

Module	Routines and Labels
BFCTL	Buffer control routines
	$BLXIO - Move block of data
	$GTBYT - Get next byte from user buffer
	$GTCWD - Get next word from user control buffer
	$GTWRD - Get next word from user buffer
	$PTBYT - Put next byte in user buffer
	$PTWRD - Put next word in user buffer
CORAL	Core buffer allocation routines
	$ALCLK - Allocate clock queue core block
	$ALOC1 - Allocate core buffer (alternate entry)
	$ALOCB - Allocate core buffer
	$ALPKT - Allocate SEND or I/O REQUEST core block
	$DEAC1 - Deallocate core buffer (alternate entry)
	$DEACB - Deallocate core buffer
	$DECLK - Deallocate clock queue core block
	$DEPKT - Deallocate SEND or I/O REQUEST core block
CRASH	Crash dump routines
	$CRASH - Crash dump routine
	$CRSBF - Internal crash stack
	$CRSBN - Starting device address
	$CRSCS - Checksum of device address
	$CRSHT - Halt to wait for the user
	$CRSUN - Crash unit number (C$$RUN) stored here
	$PANIC - Reference entry label only
CTDRV	TA11 tape cassette controller driver
	$CTINT - Controller interrupt processing
	CTINI - Controller initiator
	$CTTBL - Driver dispatch table
	SPCBK - Spacing function
	RDBLK - Read logical function

Module	Routines and Labels
CTDRV	WRBLK - Write logical function
(cont.)	WREOF - Rewind and write EOF functions
	CTOUT - Device timeout
CVRTM	Convert time routine
	$CVRTM - Convert a time interval-time units pair to a clock ticks count
DBDRV	RH11-RP04/05/06 disk pack driver
	CNTBL - Address of current unit control block
	RTTBL - Retry count for current operation
	TEMP - Temporary storage for controller number
	OFFAD - Address of current offset value
	OFFTB - Offset positioning value table
	FUNTBL - Diagnostic function table
	$DBTBL - Driver dispatch table
	DBINI - Initiator
DLDRV	RL11/RL01 disk driver
	CNTBL - Address of current unit control block
	RTTBL - Retry count for current operation
	TEMP - Temporary storage for controller number
	$DLTBL - Driver dispatch table
	DLINI - Initiator (get I/O packet)
	DLINIO - Initiate I/O operation
	DLOUT - Device timeout
	DLDIFF - Cylinder address difference calculator
	DLDVER - Error logging routine
	DLDTER - Error logging routine
DMDRV	RK611-RK06/RK07 disk cartridge driver
	CNTBL - Address off currrent unit control block
	RTTBL - Retry count for current operation
	TEMP - Temporary storage for controller number
	FUNTBL - Diagnostic function table
	OFFTB - Offset positioning data
	$DMTBL - Driver dispatch table
	DMINI - Initiator
	DMINIO - I/O initiator
	DMOUT - Device timeout
	DMECC - Error correction
	DMECOR - Memory address calculation for correction
	DMDVER - Error logging
	DMDVTO - Error logging
	DMRPAS - Controller register pass routine
	DMDINT - Diagnostic interrupt handler
DPDRV	RP11-C/E disk pack controller driver
	CNTBL - Address of current unit control block
	RTTBL - Error retry count and positioning flag
	TEMP - Temporary storage for controller number
	FUNTBL - Diagnostic function code table
	$DPTBL - Driver dispatch table
	DPINI - Initiator
	DPOUT - Device timeout
DRABO	Abort task routine
	$DRABO - Abort a specified task
DRASG	Assign a device unit to a logical unit number
	$DRASG - Assign logical unit number (LUN)

CROSS-REFERENCES

Module	Routines and Labels
DRATX	End execution of an asynchronous system trap service routine $DRATX - Asynchronous system trap (AST) service exit routine
DRCIN	Connect or disconnect an interrupt vector to an interrupt service routine (ISR) in the task's own space $DRCIN - Connect to interrupt $DISIN - Disconnect interrupt vector
DRCMT	Cancel MARK TIME and SCHEDULE REQUEST directives $DRCMT - Cancel MARK TIME requests $DRCSR - Cancel SCHEDULE requests
DRDAR	Disable or enable AST recognition directive processing $DRDAR - Disable AST recognition $DREAR - Enable AST recognitionDAR
DRDCP	Disable or enable checkpointing directive processing $DRDCP - Disable checkpointing $DRECP - Enable checkpointing
DRDSP	DRDSP contains the directive dispatch table BTRMV - Bytes to remove on exit DSPMP - Dispatch mapping table DSPTBL - Directive dispatch table $EMTRP - EMT instruction trap routine $DPLM1 - Get pointer to definition block $DPLM2 - Get size of definition block $DRATP - NOP alter priority $DRLM1 - Get first word on user stack $DRLM2 - Get first DPB word $TRTRP - TRAP instruction trap routine USRPS - Pointer to user PS word
DREIF	End execution of the issuing task directive processing $DREIF - Terminate the execution of the issuing task if the event flag is clear $DREXT - Terminate the execution of the issuing task MTQUE - Subroutine to empty queue SCNLN - Scan logical unit table
DREXP	Extend partition directive processing $DREXP - Extend the partition of the issuing task
DRGCL	Get MCR command line or release MCR command buffer directive processing $DRGCL - Get MCR command line $RLMCB - Release MCR command buffer
DRGLI	Get logical unit number information directive processing $DRGLI - Get LUN information
DRGPP	Get partition parameters directive processing $DRGPP - Get partition parameters
DRGSS	Get sense switch register contents directive processing $DRGSS - Get sense switch contents
DRGTK	Get task parameters directive processing $DRGTK - Get task parameters

CROSS-REFERENCES

Module	Routines and Labels
DRGTP	Get time parameters directive processing $DRGTP - Get current time parameters
DRMAP	Mapping and send or receive by reference directive processing $DRCRW - Create address window $DRELW - Eliminate address window $DRGMX - Get mapping context of the task $DRMAP - Map window to region $DRRRF - Receive by reference $DRSRF - Send by reference $DRUNM - Unmap address window
DRMKT	Mark time and run directive processing $DRMKT - Mark time; declare a significant event after a specified interval $DRRUN - Run the task after a specified interval or run the task after a specified interval and repeat the task periodically
DRPUT	Specify floating-point, powerfail, and receive AST traps directive processing $DRFEX - Specify floating-point exception ASTs for the issuing task $DRPUT - Specify power recovery ASTs for the issuing task $DRRCV - Specify receive ASTs for the issuing task $DRRRA - Specify receive by reference ASTs for the issuing task
DRQIO	Queue I/O directive processing ATRBK - Build attribute pointer block BDPKT - Build an I/O packet CKACC - Check if access also requested on create CKALN - Check for file already accessed on LUN CKCON - Check connect parameter buffer CKDIS - Fill disconnect parameter buffer and interlock LUN usage CKDMO - Check for volume marked for dismount CKNLN - Check for file accessed on LUN CKRAC - Check for read access priviliges and exit to transfer function CKRLK - Access or deaccess interlock CKWAC - Check for write access priviliges and exit to transfer function CKXIT - Exit polish to function exit $DQLM1 - Zero I/O status block $DQLM2 - Clear I/O status block $DRQIO - Queue I/O request $DRQIW - Queue I/O request and wait $DRQRQ - Insert I/O packet in a controller queue FCACC - Access file; check if volume marked for dismount FCCAW - Access file; check if file accessed FCCON - Connect to process FCCRE - Create file FCCTL - Function is control function FCDAC - Deaccess file; check if file accessed on LUN FCDIS - Disconnect from process FCDSP - Function code dispatch vector FCIFC - Set illegal function status FCKIL - Flush I/O queue

CROSS-REFERENCES

```
Module      Routines and Labels

DRQIO       FCNCT  - Network control function
(Cont.)     FCPKT  - Build an I/O packet
            FCRVB  - Read virtual block; check if file accessed on LUN
            FCTRN  - Function is a transfer function; address check and
                     map function
            FCWVB  - Write virtual block; check if file accessed on LUN
            FCXFR  - Insert parameter 2 (within FCCTL routine)
            FCXIT  - Clean stack and retrieve address of I/O
                     packet (prior to entry into $DRQRQ)
            FILNM  - Insert optional filename block
            IEALN  - File already accessed on LUN; set file already
                     accessed code
            IEBAD  - Bad parameter; set bad parameter status
            IEBYT  - Illegal byte count or alignment; declare odd byte
                     status
            IECMN  - Common error exit
            IEIFC  - Illegal function; declare illegal function code
                     status
            IENLN  - No file accessed on LUN; set no file accessed
                     status
            IENOD  - No buffer space available; set no buffer status
            IEOFL  - Specified device is offline
            IEOVR  - Illegal load overlay UCB; declare illegal load
                     overlay function status
            IEPRI  - Privilege violation; set privilege violation status
            IESPC  - Illegal buffer address specified; declare illegal
                     buffer status
            ISSUC  - Function is a NOOP function; declare successful
                     completion status
            MSTK   - Location to mark stack address
            MOVE3  - Move extend and access control words into I/O
                     packet
            OPPRM  - Interpret optional block address
            RQPRM  - Interpret required block address
            UNLCK  - Unlock block
            UNLKT  - Set up registers for unlock and exit
                     to control address

DRRAS       Receive and send directive processing
            $DRREC - Receive data and receive data or exit.
                     Dequeue data from the issuing task's receive
                     queue
            $DRSND - Send data; queue data in a specified task's
                     receive queue

DRREG       Attach and detach region directive processing
            $DRATR - Attach region to the current task
            $DRCRR - Create a region and optionally attach to it
            $DRDTR - Detach the specified region, unmapping if
                     necessary
            $DETRG - Detach region by attachment descriptor address

DRREQ       Request task execution directive processing
            $DRREQ - Request task execution

DRRES       Resume or suspend task execution or alter task priority
                     directive processing
            $DRATP - Alter task priority of a specified task
            $DRRES - Resume executing a task that has issued the
                     suspend directive
            $DRSPN - Suspend execution of the issuing task
```

Module	Routines and Labels
DRSED	Significant event and event flag directive processing
	$DRCEF - Clear event flag and report its polarity before clearing
	$DRDSE - Declare a significant event
	$DRRAF - Read all event flags (local and common)
	$DRSEF - Set an event flag and report its polarity before setting
	$DRWFL - Suspend task execution until LOGICAL OR of event flags occur
	$DRWFS - Suspend task execution until a specified event flag is set
	$DRWSE - Suspend execution of the issuing task until the next significant event
	$TKWSE - Execute a wait for significant event directive for the current task
DRSST	Specify SST vector directive processing
	$DRSDV - Specify debugging aid SST vector
	$DRSTV - Specify task SST vector
DTDRV	TC11 DECTAPE controller driver
	CNTBL - Address of current unit control block
	RTTBL - Error retry count and drive reset flag
	TEMP - Temporary storage for controller number
	$DTTBL - Driver dispatch table
	DTINI - Initiator
	DTCAN - Cancel I/O operation
	DTOUT - Reference label
DXDRV	RX11 floppy disk driver
	CNTBL - Address of current UCB for controller
	DXCAN - Cancel I/O entry point
	DXTBL - Driver dispatch table
	DXINI - Initiator
	DXOUT - Log device timeout
	DXPWF - Powerfail entry point
	DXRTY - Retry last function
	NXTSEC - Update block number, buffer address, and buffer pointer
	RTTBL - Error retry count for current unit
	SETBUF - Set up buffer pointer for CPU; SILO transfers
	TEMP - Temporary storage for controller number
	TRKSEC - Convert logical or physical block number to track-sector pair
ERROR	Error logging and error log processing
	$ALEB1 - Allocate an error message block (EMB); (alternate entry)
	$ALEMB - Allocate an error message block
	$BMSET - Set a driver's bit in the I/O active bitmap
	$DTOER - Log timeout error; EMB formatting routine
	DTOTMP - Device timeout storage
	$DVCER - Log device error bit errors; EMB formatting routine
	$DVERR - Same as $DVCER
	$NS0 - Call common nonsense interrupt code; group 0 - 17
	$NS1 - Call common nonsense interrupt code; group 20 - 37
	$NS2 - Call common nonsense interrupt code; group 40 - 57

CROSS-REFERENCES

Module	Routines and Labels	
ERROR (Cont.)	$NS3	Call common nonsense interrupt code; group 60 - 77
	$NS4	Call common nonsense interrupt code; group 100 - 117
	$NS5	Call common nonsense interrupt code; group 120 - 137
	$NS6	Call common nonsense interrupt code; group 140 - 157
	NSI	Nonsense interruption recursion counter
	NSIER	Nonsense interrupt errors
	OPS	Nonsense interruption old PS storage
	OPC	Nonsense interruption old PC storage
	$QEMB	Queue an error message block (EMB)
	TEMP	Nonsense interrupt PS storage
INITL	System startup and initialization routine	
	DEVMG	"DEVICE dduu: NOT IN CONFIGURATION" message
	$INITL	System gets control here after a boot to initialize and start up the system
	OPMSG	Send message to terminal
	$POOL	Start of pool
	PROMT	Terminal prompt character
	$SYBEG	Beginning of dynamic storage region
	SYSMG	System identification message
	SYSID	System ID
	$SYTOP	Last address in the Executive
	TRTRP	Non-existent memory trap routine
IOSUB	I/O related subroutine processing	
	$ACHCK	Address check, word aligned
	$ACHK2	Address check 2-byte directive parameter block
	$ACHKB	Address check byte aligned
	$ACHKP	Address check parameter block
	$ACHKW	Address check parameter block, word aligned
	$ASUMR	Assign UNIBUS mapping registers
	$BLKC1	Logical block check routine (alternate entry)
	$BLKCK	Logical block check routine
	$CEFI	Convert event flag number for I/O
	$CEFN	Convert event flag number for directive
	$CRPAS	Common register pass routine
	$DEUMR	Deassign UNIBUS mapping registers (UMRs)
	$DIV	Integer divide magnitude numbers
	$DQUMR	Dequeue from UNIBUS mapping register (UMR) wait
	$DVMSG	Device message output to task termination notification task
	$ECCOR	Common ECC correction code for RP04/RK06
	$GTPKT	Get I/O packet from request queue
	$IOALT	I/O done (alternate entry); finish I/O processing
	$IODON	I/O done; finish I/O processing
	$IOFIN	Finish I/O processing where unit and controller are not to be idle
	$IOKIL	Kill I/O; flush all I/O requests for the current task and cancel current I/O
	$LCKPR	Lock processing routine
	$MPLND	Map logical unit number (LUN); check for redirected device
	$MPLNE	Map LUN for exit
	$MPLUN	Map LUN
	$MPPHY	Map to physical address

CROSS-REFERENCES

Module	Routines and Labels
IOSUB (Cont.)	$MPPKT - Map a read/write virtual function in an I/O packet to a read/write logical function
	$MPUBM - Map UNIBUS to memory
	$MPVBM - Map virtual block number
	$MUL - Integer multiply magnitude numbers
	$RELOC - Relocate virtual address into a relocation bias and displacement in block
	$RELOM - Relocate and map address
	$RELOP - Relocate UNIBUS physical address
	$RLCH - Release channel
	$RQCH - Request channel
	$SCDV1 - Scan device tables (alternate entry)
	$SCDVT - Scan device tables
	$STMAP - Set up UNIBUS mapping address
	$WTUMR - Wait for change in UNIBUS mapping register state
LOADR	Task to load and checkpoint all nonresident tasks
	IOSB - I/O status double word
	LDRBF - R/W I/O DPB; buffer address
	LDRBK - LBN of I/O transfer
	LDRDP - R/W I/O DPB; DIC, DPB size
	LDRFC - R/W I/O DPB; function code ; LUN 1 ; EFN 1 ; I/O status doubleword address ; no AST service routine
	LDRLN - R/W I/O DPB; buffer length
	LDRTK - R/W I/O DPB; pointer to request task TCB
	$LOADR - 1. Read a non-resident task into memory and initialize it for execution 2. Read a previously checkpointed task back into memory and restart its execution 3. Write a checkpoint image of a running task and free its partition
LOWCR	START - Interrupt and trap vectors $EMTRP - EMT instruction trap $ILINS - Illegal instruction trap $IOTRP - IOT instruction trap $NONSI - Nonsense interrupt vector $TRACE - Breakpoint trap $TRP04 - Trap to 4 $TRTRP - TRAP instruction trap
	DSW - Pointers Address of directive status Directive status word FCS impure area pointer FORTRAN impure area pointer Overlay run time system impure area pointer
	$STACK - Executive stack area
LPDRV	LP11/LS11 line printer controller driver
	CNTBL - Address of UCB
	LPCAN - Cancel I/O
	LPINI - Driver initiator
	LPINT - Interruption processing
	LPOUT - Device timeout processing
	LPPWF - Powerfail return
	LPRNT - Fill line printer buffer
	$LPTBL - Driver dispatch table
	TEMP - Temporary storage for controller number

Module	Routines and Labels
MMDRV	RH11/RH70 TM02/TM03 magnetic tape controller driver
	BSPACE - Backspace one record
	CHKEOV - Check for logical end of volume
	CNTBL - Address of current UCB
	DRVCLR - Issue drive clear
	FMTBL - Format code save area
	INTADD - Current interruption service address
	LGFCN - Legal function dispatch table
	MMCAN - Cancel I/O operation
	MMDINT - TU16 diagnostic interruption and timeout handler
	MMINI - Tape controller initiator
	MMPWF - Powerfail processing
	$MMTBL - Driver dispatch table
	REWND - Rewind function
	RLCH - Release channel
	RQCH - Request channel
	RTTBL - Error retry count
	SELECT - Select drive
	SELERR - Select error
	SPCBK - Space block function
	SPCFL - Space file function
	SPTBL - Space checking
	TEMP - Temporary storage for controller number
	WRBLK - Write logical function
	WREOF - Write tape mark function
MTDRV	TM11 magnetic tape controller driver
	BSPACE - Backspace one record function
	CHKEOV - Check for logical end of volume
	CNTBL - Address of current unit control block
	INTADD - Current interuption service address
	LGFCN - Legal function dispatch table
	MTCAN - Cancel I/O operation
	$MTCLK - Reference label for timeout
	MTDINI - Diagnostic interruption and timeout handler
	MTINI - Tape controller initiator
	MTOUT - Device timeout processing
	$MTTBL - Driver dispatch table
	RDBLK - Read logical function
	REWND - Rewind function
	RTTBL - Error retry count
	SELECT - Select a tape drive
	SELERR - Select error
	SPCBK - Space block function
	SPCFL - Space file function
	SPTBL - For space checking
	TEMP - Temporary storage for controller number
	WRBLK - Write logical function
	WREOF - Write tape mark function
NLDRV	Null device driver
	$NLTBL - Driver dispatch table
	NLINI - Null driver executable code
PARTY	Memory parity interrupt handling
	ERTRK - Address/data group 0 and 1, time of last error
	EXMSG - Executive parity error message
	$PARTB - Dummy control status register (CSR) for nonexistant registers
	$MPCTL - New cache parity CSR contents

Module	Routines and Labels
PARTY	$MPCSR - Vector of cache CSR addresses
(Cont.)	MSTAT - First two parity CSRs
	PARLV - Interruption recursion level counter
	RECURS - Jump to halt processor
	STAT - Memory status register
	PARER - Memory parity error interrupt processing
PLSUB	Program logical address space (PLAS) common subroutines
	$CKACC - Check desired access of a task into a region
	$CRATT - Create attachement descriptor
	$SRATT - Search for attachment descriptor
	$SRNAM - Search for named partition
	$SRWND - Search for specified address window
	$UNMAP - Unmap address window
POWER	Power failure recovery processing
	$LDPWF - Save APR5; reference label for LOAD
	PDOWN - Powerfail interrupt processing
	$POWER - Power failure recovery processing routine
	PUP - Power up interrupt processing
	PWBTM - Volatile register storage
	PWVCT - Powerfail vector
PPTAB	Device tables
	$PPDAT - Start of device tables
	$PPEND - End of device tables
PRDRV	PC11/PR11 paper tape reader driver
	CNTBL - Address of unit control block
	PRCAN - In process I/O tranfers are not terminated
	PRINI - Controller initiator
	$PRINT - Controller interruption processing
	PROUT - Device timeout processing
	PRPWF - Powerfail return
	$PRTBL - Driver dispatch table
	TEMP - Temporary storage for controller number
QUEUE	General queue manipulation processing
	$CLINS - Clock queue insertion
	$CLRMV - Clock queue removal
	$QINSF - Queue insertion at end of list
	$QINSP - Queue insertion by priority
	$QMCRL - Queue MCR command line
	$QRMVF - Queue removal from front of list
	$QRMVT - Queue removal by TCB address
REQSB	Task request related subroutines
	$ABCTK - Abort current task
	$ABTSK - Abort task
	$ACTTK - Put task in active task list
	$ACTRM - Remove task from the active task list
	$BILDS - Build stack and initialize header
	$CHKPT - Checkpoint task
	$DASTT - Declare AST trap
	$DQAC - Dequeue AST block queued by $QASTC
	$EXRQF - Executive request with FIFO queue insert
	$EXRQN - Executive request with no queue insertion
	$EXRQP - Executive request with queue insert by priority
	$FNDSP - Find space in PCB list
	$ICHKP - Initiate checkpoint
	$LOADT - Put task in loader queue

Module	Routines and Labels
REQSB (Cont.)	$MAPTK - Map task address window
	$NXTSK - Assign next task to partition
	$QASTC - Queue AST to task
	$QASTT - Queue AST to task
	$RLPAR - Release task partition; get PCB address
	$RLPR1 - Release partition; clear busy
	$SETCR - Set conditional schedule request
	$SETF - Set event flag; convert to mask and address
	$SETM - Set event flag
	$SETRQ - Set schedule request
	$SETRT - Set schedule request for current task
	$STPCT - Stop current task
	$STPTK - Stop task
	$SRSTD - Search system task directory
	$TSTCP - Test if checkpoint should be initiated
	$TSKRP - Task request (default UIC)
	$TSKRQ - Task request (UCB specified)
	$TSKRT - Task request (default UCB)
	$UISET - Establish default UIC and current UIC
SSTSR	Synchronous system trap (SST) service routine processing
	$EMSST - Non-RSX EMT/TRAP instruciton
	FLFRK - Floating-point fork block
	FLSTS - Floating-point status
	$FLTRP - Floating-point exception (11/40)
	$FLTRP - Floating-point exception (11/45)
	$FPINT - Programmed interrupt request processing
	$ILINS - Illegal or reserved instruction trap routine
	$IOTRP - IOT instruction trap routine
	$SGFLT - Segment fault trap routine
	SSTXT - Common SST exit routine
	$TRACE - TRACE (T-bit) or break point instruction (BPT) trap routine
	$TRP04 - Traps occuring at 4 (odd address, non-existant memory, etc.) trap routine
SYSCM	System common data areas
	$ABTIM - H.CSP; current stack pointer
	$ACTHD - T.MXSZ; active task list listhead
	$BTMSK - Bit mask table
	$CFLPT - W.BOFF; pointer to first checkpoint file PCB
	$CKCNT - T.ASTL+2; address of clock count register
	$CKCSR - T.EFLG; address of clock control status register
	$CKLDC - T.EFLG+2; clock load count
	$CLKHD - P.TCB; clock queue
	$COMEF - T.TCB; common event flags 1. - 16.
	$COPT - P.STAT; pointer to command output UCB
	$CRAVL - P.PRI; dynamic storage listhead
	$DEVHD - H.FCS; pointer to first DCB
	$DYPMN - H.VEXT; February, March
	$ERRHD - Error logging message queue listhead
	$ERRLM - Limit on resident error logging data
	$ERRPT - W.BHVR; pointer to error logger TCB
	$ERRSQ - Universal error sequence number
	$ERRSV - Pointer to error file identification
	$ERRSZ - Resident bytes of error logging data
	$EXSIZ - W.BSIZ; address of last byte in Executive
	$FMASK - P.WAIT+2; system feature mask
	$FRKHD - P.SIZE; fork queue listhead

CROSS-REFERENCES

```
Module     Routines and Labels

SYSCM      $HEADR - T.LNK; pointer to current task header
(Cont.)    $INTCT - P.MAIN; clock interrupt ticks count
           $IOABM - Device I/O active bitmap
           $LDRPT - H.FORT; pointer to loader TCB
           $LOGHD - T.NRPC; logical device assignment list
           $LSTLK - T.LDV; lock word (TCB address of owner)
           $MCRCB - T.LBN+1; MCR command block address
           $MCRPT - H.OVLY; pointer to MCR TCB
           $MXEXT - GLobal task size limit for extend task
                    directive
           $PARHD - H.CUIC; pointer to partition table
           $PARPT - P.BUSY; parity address vector table pointer
           $PKAVL - Pointer to first packet in list
           $PKMAX - Maximum number allowed in list
           $PKNUM - Number of packets currently in list
           $PWRFL - H.EFLM; powerfail recovery pointer
           $RQSCH - H.EFSV; schedule request TCB address
           $SHFPT - W.BATT; pointer to shuffler TCB
           $SIGFL - H.EFLM+2; task waiting for significant event
           $STKDP - H.FPSA; stack depth indicator
           $SYSID - T.NAM+2,T.RCVL; system identification
           $SYSIZ - Size of memory in 32 word blocks
           $SYUIC - T.UCB; default system UIC (mapped or unmapped)
           $TKNPT - T.RCVL+2; pointer to TKTN TCB
           $TKPS  - Ticks per second
           $TKTCB - H.FPVA; pointer to current task TCB
           $TSKHD - W.BLVR; pointer to system task directory
           $TTNS  - Tick of second
           $UMRHD - Mapping assignment block listhead
           $UMRWT - UMR wait queue listhead

SYSDF      Contains directive status codes, system global and
             control block offset definitions

SYSTB      Contains system device tables
           $DEVTB - Device tables

SYSXT      System entrance, exit, and processor dispatching
                    routines
           $DIRSV - Directive save routine
           $DIRXT - Directive exit
           $FINBF - Finish terminal input buffered I/O
           $FORK0 - Fork and create system process (alternate
                    entry)
           $FORK1 - Fork and create system process and save R5
           $FORK2 - Fork and create system process (CINT$
                    directive)
           $FORK  - Fork and create system process (called from
                    I/O driver)
           $INTSC - Interrupt save (CINT$ directive)
           $INTSE - Interrupt save (error logging devices)
           $INTSV - Interrupt save
           $INTX1 - Interrupt exit
           $INTXT - Interrupt exit
           $NONSI - Nonsense interrupt RTI routine
           $NS0   - Nonsense interrupt vector
           $NS1   - Nonsense interrupt vector
           $NS2   - Nonsense interrupt vector
           $NS3   - Nonsense interrupt vector
           $NS4   - Nonsense interrupt vector
           $NS5   - Nonsense interrupt vector
```

Module	Routines and Labels
SYSXT (Cont.)	$NS6 - Nonsense interrupt vector
	$NS7 - Nonsense interrupt vector
	RESCH - Rescheduling requested; clear schedule request
	$SAVNR - Save non-volatile registers
	$SWSTK - Switch from task stack to system stack
SYTAB	System tables needed for resident tasks and bootstrapping the system
	.LDRHD - Loader task header
	$PCBS - Loader partition control block
	$STD - Loader task control block
TDSCH	Time dependent scheduling and device timeout processing
	$CKINT - Clock interrupt processing routine
	DVOUT - Test for one second elapsed time
	ROBIN - Executive round robin scheduling
	RNDCT - Clock ticks to next schedule interval
	SWAP - Disk swapping algorithm; reduce swapping priority of resident tasks
	SWPCT - Clock ticks to next swapping interval
	TDS - Time dependent scheduling
	TIMXT - Exit time dependent scheduling if no unprocessed clock ticks remain
	UPTIM - Update absolute and real time of day and date
TTDRV	Terminal driver for DL11-A line interface and DH11/DJ11/DZ11 line multiplexers
	BECHOB - Echo next byte
	CLKSW - DM11-BB clock switch word
	CNTBL - Address of unit control block
	CRTRUB - Backspace, space, backspace (/ /)
	CTRLC - Control output message (MCR>)
	CTRLR - Control R processing
	CTRLU - Control output message (U)
	CTRLZ - Control output message (Z)
	DHCSR - DH11 CSR address
	$DHINP - DH11 terminal multiplexer input interrupt processing
	$DHOUT - DH11 terminal multiplexer output interrupt processing
	DHTBL - Pointer to DH11 table
	DHTMP - Temporary storage for controller number (DB11)
	DJCSR - DJ11 CSR address
	$DJINP - DJ11 terminal multiplexer input interrupt processing
	$DJOUT - DJ11 terminal multiplexer output interrupt processing
	DJSAV - DJ11 terminal multiplexer save routine
	DJTBL - Pointer to DJ11 UCB table
	DJTMP - Temporary storage for controller number (DJ11)
	$DLINP - DL11 terminal input interrupt processing
	$DLOUT - DL11 terminal output interrupt processing
	DLSAV - DL11 terminal save routine
	DLTMP - Temporary storage for controller number (DL11)
	$DM11B - DM11-B or DM11-BB modem control interrupt processing
	DMHUP - Subroutine to hang up a DM11-BB unit if not ready
	DMSAV - DH11 terminal multiplexer save routine
	DMTBL - DM11-BB CSR address

CROSS-REFERENCES

Module	Routines and Labels
TTDRV (Cont.)	DMTMO - DM11-BB time out routine
	DOCTLC - Lock out input characters
	DZCLK - Clock queue entry address
	DZCLKS - DZ11 clock switch word
	DZCSR - DZ11 CSR address
	$DZINP - DZ11 terminal multiplexer input interrupt processing
	$DZOUT - DZ11 terminal multiplexer output interrupt processing
	DZSAV - DZ11 terminal multiplexer save routine
	DZTBL - Pointer to DZ11 UCB table (indexed by controller number)
	DZTMO - DZ11 time out routine
	DZTMP - Temporary storage for controller number (DZ11)
	ECHOB1 - Display (send out) a character
	ECHOB - Echo next byte
	FCHAR - Send a character to a terminal
	FILTB - LA30S carriage return fill table
	FWRITE - Breakthrough write (disallow control-O)
	GETBF - Get input buffer and set terminal control block
	GETBF2 - Allocate a core block
	GMCTAB - Terminal characteristics
	ICHAR - Process an input character
	INPINI - Copy UCB address
	INPPT - Input request in progress
	INPT0 - Enable input character handling
	INPT1 - Fork to finish an input request
	INPT2 - End-of-line fork process
	IODON - Finish I/O operation
	JTTINI - Go to terminal initiator
	LEVHSM - Leave hold-screen mode ()
	MECHO1 - Multi-echo processing
	OCHAR - Send a character to a terminal
	OUTPT - Start or continue an output stream
	OUTPT1 - Test if a fill should be echoed
	SYNTAB - Escape sequence syntax table
	TCHR1 - Teminal characteristics
	TINP1 - Unsolicited input fork process
	TICAN - Cancel I/O operation (force I/O complete)
	TTCHK - Terminal driver special parameter checking
	TTHUP - Cancel I/O and BYE if DM11 or DZ11 line hangs up
	TTINI - Terminal initiator
	TTOUT - Terminal driver timeout entry point
	TTOUT1 - Terminal timeout; finish I/O operation
	TTPWF - Powerfail entry point (loaded as a loadable driver)
	$TTTBL - Device initiator entry point
	UCBTB - Address of UCB for line
	UCJTB - Address of line's UCB (DJ11)
	UCZTB - UCB address for each line; indexed by line (unit) number
XBDRV	DA11-B Interprocessor Communications Driver
	CNTBL - Unit impure data table (reference label)
	UNITBL - Unit impure data table
	$XBTBL:: - Device dispatch table
	DBINIT - DA11-B parallel communications link controller I/O initiator entry point
	SUCC: - Successful completion entry
	DBTMO: - Device timeout entry

CROSS-REFERENCES

Module	Routines and Labels	
XBDRV (Cont.)	DBCANC:	- Device cancellation entry
	DBPWRF:	- Device powerfail entry
	$XBINT::	- DA11-B interrupt routine
	IEVER:	- Unrecoverable error finish
	SUCDN:	- Successful completion
	INTDN:	- Finish I/O
	TXDN:	- Check for enable receive
	NOTHING:	- Check for interrupt
	EXIT:	- Wait for transfer completion
	ERR:	- Error checking
	DBSET:	- Check for valid UCB address and device online
	DBEXIT:	- Exit from interrupt
	XBRCV:	- Initiate device for unsolicited receive
	RESYNC:	- Receive error. Check for receiver active
	MYSYNC:	- Receive error. Device must be resynced
XMDRV	RSX-11M DMC11 Driver	
		UCB displacements used
	U.XQ:	- Transmit listhead address
	U.RQ:	- Receive listhead address
	U.ERR:	- Error status
	U.XAST:	- Exception AST block address
	U.ABO:	- Number of I/O requests marked for abort
		Listhead displacements
	IOTYPE:	- SEL0 request type
	COUNT:	- Sent count
	UMR:	- Next UMR to use
	UMRSUM:	- Sum of both UMR addresses
	BASE:	- Microprocessor base table
	$XMTBL::	- Driver dispatch table
	CNTBL:	- UCB address table
	TEMP:	- Temporary unit save
	LIST:	- Listheads
	BTAB:	- Multiple unit base table addresses
	XLTAB:	- Multiple unit listhead addresses
	AXTAB:	- Address extension bits
	XMRET:	- Return point
	XMINIT:	- Initiate DMC I/O entry
	PKTOK:	- Finish packet with "IS.SUC" status
	FINPKT:	- Alternate entry for PKTOK
	XMDNR:	- Device not ready
	XMSPC:	- Illegal buffer
	XMIFC:	- Illegal function code
	TRAN:	- Transfer function
	TRY:	- Try to give transfer request to DMC
	GIVE:	- Give buffer address and count to DMC
	SETDMC:	- Initialize DMC hardware
	WRDYIC:	- Release port and wait for RDYI clear
	WRDYIS:	- Wait for RDYI set
	MAP22:	- Do 22-bit mapping and load data port
	$XMINP::	- Process RDYI interrupt
	XMTRDY:	- DMC ready for transmit buffer
	RCVRDY:	- DMC ready for receive buffer
	FPERR:	- Force procedure error
	$XMOUT::	- Process completion interrupt
	XMTCOM:	- Transmit buffer complete
	RCVCOM:	- Receive buffer complete
	CNTLO:	- Control out
	BAD:	- Exit from interrupt (reference label)
	XMINTX:	- Exit from interrupt
	XMSET:	- Setup for interrupt routines and transfer

CROSS-REFERENCES

Module	Routines and Labels	
XMDRV (Cont.)	RDYINT:	- Give buffer to DMC
	BUFCOM:	- Process buffer complete
	XMFRK:	- Start fork process if fork process not running
	XMTMO:	- Timeout processing entry
	XMCANC:	- Cancel I/O entry
	XMPWRF:	- Powerfail recovery entry
	XMCANC:	- Indicate I/O kill was done
	KILL:	- Kill the device
	RQPABO:	- Alternate entry for KILL:
	RQP:	- Alternate entry for KILL:
	QXAST:	- Queue an AST to the CCP (clear AST block address)
	QAST:	- Queue an AST to the CCP (declare significant event)
	DQRAP:	- Dequeue and return an I/O packet
	RAP:	- Return an already dequeued I/O packet
	ITRY:	- Try to set up another buffer from interrupt level
	PORT:	- Request the data port
	IPORT:	- Request the data port from interrupt level
	NTXMT:	- Initiate DECNET transmit
	NTRCV:	- Initiate DECNET receive
	NTFIX:	- DECNET error recovery
	NTABO:	- Abort by CCB address
XPDRV	DP11 Synchronous Communications Driver	
	CNTBL:	- Reference label for UNITBL:
	UNITBL:	- UCB addresses
	TEMP:	- Reference label for UNIT:
	UNIT:	- Temporary storage for unit number
	$XPTBL::	- Driver dispatch table
	DPINIT:	- DP11 Synchronous Communications Controller I/O initiator
	DPSUCC:	- Return successful status
	DPFIN:	- Idle controller and mark unit idle
	DPPWRF:	- Power fail service routine
	DPCANC:	- I/O cancellation entry
	DPTMO:	- Timeout service routine
	$XPINP::	- DP11 input interrupt service routine
	$XPOUT::	- Transmitter interrupt service routine
	DPTXND:	- End of transmission
	DPRXER:	- Receiver error detected. Resync controller
	DPRCV:	- Unsolicited receive initialization. Activate controller
	DPSET:	- Set up register R4 with CSR address, R5 with UCB address, unit number in low-order 4 bits of unit
	DPSXT:	- Dismiss interrupt
XQDRV	DQ11 Synchronous Communications Driver	
	CNTBL:	- Reference label for UNITBL:
	UNITBL:	- UCB address tabel
	TEMP:	- Reference label for UNIT:
	UNIT:	- Temporary storage for unit number
	$XQTBL::	- Device dispatch table
	DQINIT:	- DQ11 Synchronous Communications Controller I/O initiator
	DQRET:	- Return from initiator
	SUCC:	- Do I/O done (successful)
	UNSUCC:	- Do I/O done (unsuccessful)
	DQCANC:	- I/O cancellation entry
	DQPWRF:	- Powerfail routine entry
	DQTMO:	- Timeout routine entry

Module	Routines and Labels	
XQDRV (Cont.)	TNEXT:	- Select appropriate buffer address register to use (transmit)
	TOAN:	- Select appropriate buffer address register to use (transmit - double buffering ahead)
	RNEXT:	- Select appropriate buffer address register to use (receive)
	ROAN:	- Select appropriate buffer address register to use (receive - double buffering ahead)
	$XQOUT::	- Transmit interrupt routine
	TXTRN:	- Finish transmit
	DQTXDN:	- Transmit done
	ERROR:	- Error routine
	RBCERR:	- Receive BCC error
	DQSET:	- Set up R4 with RXCSR address, R5 with UCB address, unit number in low-order four bits of unit.
	$DQEXIT:	- Jump to $INIXT to dismiss interrupt
	$XQINP::	- Receive interrupt routine
	RXENT:	- End of receive routine. Clear spurious clock error, save registers, check for buffers finished.
	DQRCV:	- Initialize DQ for unsolicited receive
	RESYNC:	- Re-sync or initialize receiver to receive from a dead start.
	SETUP:	- Set up device with next receive buffer
XUDRV	DU11 Synchronous Communications Driver	
	CNTBL:	- Reference label (for UNITBL:)
	UNITBL:	- UCB address table
	TEMP:	- Reference label (for UNIT:)
	$XUTBL::	- Driver dispatch table
	DUINIT:	- DU11 Synchronous Communication Controller I/O initiator
	DUSUCC:	- Return successful status for mode change full- or half-duplex)
	DUFIN:	- End mode change request routine
	DUPWRF:	- Powerfail service routine
	DUCANC:	- I/O cancellation service routine
	DUTMO:	- Timeout service routine
	$XUINP::	- Input interrupt service routine
	$XUOUT::	- Transmitter interrupt service routine
	DUTXND:	- End of transmitter interrupt service
	DURXER:	- Receiver error - re-sync controller
	DURCV:	- Activate controller to be ready to accept data
	DUSET:	- Set up R4 with CSR address, R5 with UCB address, unit number in low-order 4 bits of unit
	DUSXT:	- Jump to $INTXT to dismiss interrupt

Module	Routines and Labels	
XWDRV	DUP11 Synchronous Communications Driver	
	CNTBL:	- Reference label for UNITBL:
	UNITBL:	- UCB address table
	TEMP:	- Reference label for UNIT:
	UNIT:	- Temporary storage for unit number
	$XWTBL::	- Driver dispatch table
	DWINIT:	- DUP11 Synchronous Communication Controller I/O initiator
	DWSUCC:	- Successful device mode change request
	DWFIN:	- End of device mode change request routine
	DUPWRF:	- Powerfail service routine
	DWCANC:	- I/O cancellation service routine
	DWTMO:	- Timeout service routine
	$XWINP::	- DUP11 input interrupt service routine
	$XWOUT::	- Transmitter interrupt service routine
	DWTXND:	- End of transmitter interrupt service
	DWRXER:	- Receiver error detected. Resync controller
	DWRCV:	- Activate controller to be ready to receive data
	DWSET:	- Set R4 with CSR address, R5 with UCB address, unit number in low-order 4 bits of unit

9.2 RSX-11M EXECUTIVE GLOBAL CROSS-REFERENCE

The Executive global cross-reference contains an alphabetic listing of each global symbol along with its value and the name of each referencing module.

The value contains the suffix -R if the symbol is relocatable.

The symbol # preceeds the module in which the symbol is defined.

Symbol	Value	Modules That Reference Symbol				
C.SYST	000006	# EXEDF	TTDRV			
DV.MXD	000100	# EXEDF	IOSUB			
DV.PSE	010000	# EXEDF	INITL	IOSUB		
DV.TTY	000004	# EXEDF	IOSUB			
DV.UMD	000200	DRQIO	# EXEDF	IOSUB		
D$$YNM	000000	# SYSDF				
D.DSP	000012	DRQIO	# EXEDF	IOSUB	POWER	TDSCH
D.MSK	000014	DRQIO	# EXEDF			
D.NAM	000004	DRASG	# EXEDF	INITL	IOSUB	
D.PCB	000034	DRQIO	# EXEDF	IOSUB	POWER	TDSCH
D.RS00	000000	DRSED	# SYSDF			
D.RS1	177777	CORAL	DREXP	DRMAP	DRREG	DRREQ
		# SYSDF				
D.RS10	177766	DRDCP	# SYSDF			
D.RS16	177760	DRDSP	DRMAP	DRMKT	DRREG	DRRES
		PLSUB	# SYSDF			
D.RS17	177757	# SYSDF				
D.RS19	177755	# SYSDF				
D.RS2	177776	DRDSP	DRGPP	DRMAP	DRRAS	# SYSDF
D.RS22	000002	DREIF	DRRES	DRSED	# SYSDF	
D.RS5	177773	DRGLI	DRQIO	# SYSDF		
D.RS6	177772	DRQIO	# SYSDF			
D.RS7	177771	DRABO	DRREQ	DRRES	# SYSDF	
D.RS8	177770	DRDAR	DRDCP	DREXP	DRMAP	DRPUT
		DRRAS	DRRES	# SYSDF		

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

D.RS80    177660      DRATX     DRGCL     DRPUT   # SYSDF
D.RS81    177657    # SYSDF
D.RS84    177654      DREXP     DRMAP   # SYSDF
D.RS85    177653      DRMAP   # SYSDF
D.RS86    177652      DRMAP     DRREG     PLSUB   # SYSDF
D.RS87    177651      PLSUB   # SYSDF
D.RS90    177646      DRASG   # SYSDF
D.RS92    177644      DRASG   # SYSDF
D.RS93    177643      CVRTM   # SYSDF
D.RS94    177642      DRREG   # SYSDF
D.RS95    177641      DRRES   # SYSDF
D.RS96    177640      IOSUB   # SYSDF
D.RS97    177637      DRDSP     DRSED     IOSUB   # SYSDF
D.RS98    177636      DRSST     IOSUB     SSTSR   # SYSDF
D.RS99    177635      DRDSP   # SYSDF
D.UCB     000002      DRASG   # EXEDF     INITL     IOSUB
D.UCBL    000010      DRASG   # EXEDF     INITL     IOSUB
D.UNIT    000006      DRASG   # EXEDF     INITL     IOSUB
D.VCAN    000002    # EXEDF     IOSUB
D.VINI    000000      DRQIO   # EXEDF
D.VOUT    000004    # EXEDF     TDSCH
D.VPWF    000006    # EXEDF     POWER
EC.DTO    000140      ERROR   # EXEDF
EC.DVC    000001      ERROR   # EXEDF
EC.NSI    000141      ERROR   # EXEDF
E.LGTH    000056      ERROR   # EXEDF
E.OPC     000022      ERROR   # EXEDF
E.RTRY    000016    # EXEDF     IOSUB
IE.ABO    177761      DTDRV     IOSUB     LPDRV     TTDRV
IE.ALN    177736      DRQIO
IE.BAD    177777      ADDRV     DRQIO     ICDRV     TTDRV
IE.BBE    177710      DRDRV     MMDRV
IE.BLK    177754      DXDRV     IOSUB
IE.BYT    177755      DRQIO     ICDRV
IE.CNR    177667      GRDRV
IE.DAA    177770      IOSUB
IE.DAO    177763      MMDRV
IE.DNA    177771      IOSUB
IE.DNR    177775      ADDRV     DBDRV     DKDRV     DMDRV     DRDRV
                      ICDRV     MMDRV     TDSCH     TTDRV
IE.EOF    177766      MMDRV     NLDRV     TTDRV
IE.EOT    177702      MMDRV
IE.EOV    177765      MMDRV
IE.FHE    177705      MMDRV
IE.FLN    177657      ICDRV
IE.IEF    177637      GRDRV
IE.IFC    177776      DBDRV     DKDRV     DMDRV     DRDRV     DRQIO
                      DTDRV     ICDRV     IOSUB     ISDRV     MMDRV
                      TTDRV     UDDRV
IE.LCK    177745      IOSUB
IE.MOD    177753      ICDRV     ISDRV
IE.NLN    177733      DRQIO
IE.NOD    177751      DRQIO     IOSUB     TTDRV
IE.OFL    177677      DRQIO
IE.OVR    177756      DRQIO
IE.PRI    177760      DRQIO     ICDRV     IOSUB     TTDRV
IE.RSU    177757      TTDRV
IE.SPC    177772      ADDRV     DRQIO     GRDRV     ICDRV     MMDRV
                      TTDRV
IE.ULK    177653      IOSUB
IE.VER    177774      DBDRV     DKDRV     DMDRV     DRDRV     DTDRV
                      DXDRV     MMDRV
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

IE.WCK    177652      DBDRV     DKDRV     DMDRV     DRDRV
IE.WLK    177764      DBDRV     DKDRV     DMDRV     DRDRV     DTDRV
                      MMDRV
IO.ATT    001400      DRQIO     IOSUB     TTDRV
IO.CLN    003400      DREIF
IO.CON    015400      GRDRV
IO.DET    002000      DREIF     DRQIO     IOSUB     TTDRV
IO.DIS    016000      GRDRV
IO.EOF    003000      MMDRV
IO.FLN    012400      ICDRV
IO.GTS    002400      TTDRV
IO.LOV    001010      DRQIO
IO.ONL    017400      ICDRV
IO.RLB    001000      DBDRV     DKDRV     DMDRV     DRDRV     DRQIO
                      DTDRV     DXDRV     LOADR     MMDRV     TTDRV
IO.RLV    001100      DTDRV     MMDRV
IO.RPR    004400      TTDRV
IO.RVB    010400      DRQIO     IOSUB
IO.STC    002500      MMDRV
IO.STP    016400      GRDRV
IO.ULK    005000      IOSUB
IO.WLB    000400      DMDRV     DRDRV     DRQIO     DXDRV     IOSUB
                      LOADR     MMDRV     NLDRV     TTDRV
IO.WLC    000420      DBDRV     DKDRV     DMDRV     DRDRV
IO.WLT    000410      DMDRV     DRDRV
IO.WLV    000500      DTDRV
IO.WVB    011000      IOSUB
IQ.UMD    000004      DRQIO     ERROR     IOSUB
IQ.X      000001      DBDRV     DKDRV     DMDRV     DRDRV     DXDRV
IS.RDD    000002      DXDRV
IS.SUC    000001      ADDRV     DBDRV     DKDRV     DMDRV     DRDRV
                      DRQIO     DTDRV     DXDRV     GRDRV     ICDRV
                      IOSUB     ISDRV     LPDRV     MMDRV     NLDRV
                      TTDRV     UDDRV
I$$S11    000000      ISDRV
I.FCN     000012    # EXEDF     GRDRV     ICDRV     NLDRV     TTDRV
I.PRI     000002    # EXEDF     TTDRV
I.PRM     000024    # EXEDF     GRDRV     ICDRV     NLDRV     TTDRV
I.TCB     000004    # EXEDF     GRDRV     ICDRV     TTDRV
KISAR5    172352    # EXEDF     QUEUE
KISAR6    172354      DRMAP   # EXEDF     GRDRV     TTDRV
L.ASG     000010      DRASG   # EXEDF
L.NAM     000002      DRASG   # EXEDF
L.TYPE    000005      DRASG   # EXEDF
L.UCB     000006      DRASG   # EXEDF
L.UNIT    000004      DRASG   # EXEDF
M$$EXT    000000    # SYSDF
M$$MGE    000000    # SYSDF
M.BFVH    000011    # EXEDF     IOSUB
M.BFVL    000012    # EXEDF     IOSUB
M.LGTH    000014    # EXEDF     IOSUB
M.UMRA    000002    # EXEDF     IOSUB
M.UMRN    000004    # EXEDF     IOSUB
M.UMVH    000010    # EXEDF     IOSUB
M.UMVL    000006    # EXEDF     IOSUB
PR4       000200    # EXEDF     GRDRV
PR5       000240    # EXEDF     TTDRV
PR6       000300    # EXEDF     ICDRV
PR7       000340    # EXEDF     GRDRV
PS        177776    # EXEDF     GRDRV     ICDRV     TTDRV
P.ATT     000036      DRREG     PLSUB   # SYSDF
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

P.BLKS    000016    # EXEDF   # SYSDF
P.BUSY    000024    # EXEDF   # SYSDF
P.HDR     000032      DREXP     DRRES     ERROR     LOADR     REQSB
                    # SYSDF     SYSXT     TDSCH     TTDRV
P.IOC     000003    # EXEDF   # SYSDF
P.LGTH    000042      DREIF     DRREG     REQSB   # SYSDF
P.LNK     000000    # EXEDF   # SYSDF
P.MAIN    000012    # EXEDF   # SYSDF
P.NAM     000004    # EXEDF   # SYSDF
P.OWN     000026    # EXEDF   # SYSDF
P.SIZE    000016    # EXEDF   # SYSDF
P.STAT    000030    # EXEDF   # SYSDF
P.SUB     000010    # EXEDF   # SYSDF
P.SWSZ    000022    # EXEDF   # SYSDF
P.TCB     000026    # EXEDF   # SYSDF
P.WAIT    000020    # EXEDF   # SYSDF
SP.EIP    000001      ERROR   # EXEDF     IOSUB     SYSXT
SP.ENB    000002      ERROR   # EXEDF
S$$IEN    000115    # SYSCM
S$$LDC    000001    # SYSCM
S$$RTZ    000074    # SYSCM
S$$TPS    000074    # SYSCM
S.BMSK    177776      DTDRV     ERROR   # EXEDF   # SYSDF     SYSXT
S.BMSV    177774      ERROR   # EXEDF     IOSUB   # SYSDF     SYSXT
S.CCB     000030    # SYSDF
S.CON     000010      DBDRV     DKDRV     DMDRV     DRDRV     DXDRV
                    # EXEDF     IOSUB     MMDRV     POWER   # SYSDF
                      SYSXT     TDSCH
S.CSR     000012      ADDRV     DBDRV     DKDRV     DMDRV     DRDRV
                      DTDRV     DXDRV     ERROR   # EXEDF     GRDRV
                      ICDRV     INITL     LPDRV     MMDRV   # SYSDF
                      TDSCH     TTDRV
S.CTM     000006      DBDRV     DKDRV     DMDRV     DRDRV     DTDRV
                      DXDRV   # EXEDF     LPDRV     MMDRV   # SYSDF
                      SYSXT     TDSCH     TTDRV
S.DHCK    000030    # SYSTB     TTDRV
S.FLG     000000      ICDRV
S.FRK     000016    # EXEDF     IOSUB   # SYSDF     SYSXT
S.ITM     000007      DBDRV     DKDRV     DMDRV     DRDRV     DTDRV
                      DXDRV   # EXEDF     LPDRV     MMDRV   # SYSDF
                      TTDRV
S.LHD     000000    # EXEDF   # SYSDF
S.MPR     000030      IOSUB   # SYSDF
S.PKT     000014      DBDRV     DKDRV     DMDRV     DRDRV     DTDRV
                      DXDRV     ERROR   # EXEDF     IOSUB     LPDRV
                      MMDRV   # SYSDF     TTDRV
S.PRI     000004      ERROR   # EXEDF     IOSUB   # SYSDF     SYSXT
                      TDSCH     TTDRV
S.RCNT    177772      ERROR   # EXEDF     IOSUB   # SYSDF
S.ROFF    177773      ERROR   # EXEDF     IOSUB   # SYSDF
S.STS     000011      DBDRV     DKDRV     DMDRV     DRDRV     DTDRV
                    # EXEDF     IOSUB     LPDRV     MMDRV   # SYSDF
                      TTDRV
S.VCT     000005    # EXEDF   # SYSDF
TS.CKR    000100    # EXEDF     TTDRV
TS.RDN    040000    # EXEDF     TTDRV
T.ASTL    000016    # EXEDF     GRDRV     TTDRV
T.EXT     000000    # SYSDF
T.IOC     000003    # EXEDF     GRDRV     TTDRV
T.LGTH    000070    # SYSDF
T.PCB     000046    # EXEDF     TTDRV
T.STAT    000032    # EXEDF     TTDRV
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

T.ST2     000034    # EXEDF     GRDRV     TTDRV
T.ST3     000036    # EXEDF     ICDRV     TTDRV
T2.ABO    000100    # EXEDF     GRDRV
T2.AST    100000    # EXEDF     GRDRV     TTDRV
UC.KIL    000004    # EXEDF     IOSUB
UC.LGH    000003      DRQIO   # EXEDF
UC.NPR    000100    # EXEDF     IOSUB
UC.PWF    000020    # EXEDF     POWER
UC.QUE    000040      DRQIO   # EXEDF
UISAR0    177640      DRMAP   # EXEDF
UISDR0    177600      DRMAP   # EXEDF
US.BSY    000200    # EXEDF     IOSUB     TTDRV
US.CRW    000004    # EXEDF     TTDRV
US.DSB    000010    # EXEDF     TTDRV
US.ECH    000002    # EXEDF     TTDRV
US.FOR    000040      DRQIO     DRRES   # EXEDF     IOSUB
US.LAB    000004      DRQIO   # EXEDF     MMDRV
US.MDM    000020      DRQIO   # EXEDF
US.MNT    000100      DRQIO     DRRES   # EXEDF     IOSUB
US.OFL    000001      DRQIO   # EXEDF     INITL     POWER     TTDRV
US.OUT    000001    # EXEDF     TTDRV
US.PUB    000004      DRQIO   # EXEDF
US.SPU    000002      DBDRV     DKDRV     DMDRV     DRDRV   # EXEDF
US.UMD    000010      DRQIO   # EXEDF     IOSUB
US.WCK    000010      DBDRV     DKDRV     DMDRV     DRDRV   # EXEDF
U.ACP     000032      DRRES   # EXEDF     IOSUB
U.ATT     000022      DRASG     DREIF   # EXEDF     IOSUB
                      TTDRV
U.BUF     000024      ADDRV     BFCTL     DBDRV     DKDRV     DMDRV
                      DRDRV     DTDRV     DXDRV   # EXEDF     IOSUB
                      LPDRV     MMDRV     TTDRV
U.CNT     000030      ADDRV     DBDRV     DKDRV     DMDRV     DRDRV
                      DTDRV     DXDRV   # EXEDF     IOSUB     LPDRV
                      MMDRV     TTDRV
U.CTL     000004      DRQIO   # EXEDF     IOSUB     POWER
U.CW1     000010      DRGLI     DRQIO     DRRES   # EXEDF     INITL
                      IOSUB
U.CW2     000012      ADDRV     DBDRV     DMDRV     DTDRV   # EXEDF
                      IOSUB     LPDRV     MMDRV     TTDRV
U.CW3     000014      DMDRV   # EXEDF     IOSUB     MMDRV     TTDRV
U.CW4     000016    # EXEDF     LPDRV     TTDRV
U.DMCS    000064    # SYSTB     TTDRV
U.OWN     177776      DRQIO   # EXEDF
U.RED     000002    # EXEDF     IOSUB
U.SCB     000020      DBDRV     DKDRV     DMDRV     DRDRV     DRQIO
                      DRRES     DTDRV     DXDRV   # EXEDF     ICDRV
                      INITL     IOSUB     LPDRV     MMDRV     SYSXT
                      TTDRV
U.STS     000005      DBDRV     DKDRV     DMDRV     DRDRV     DRQIO
                      DRRES   # EXEDF     IOSUB     MMDRV     POWER
                      TDSCH     TTDRV
U.ST2     000007      DRQIO   # EXEDF     INITL     IOSUB     POWER
                      TTDRV
U.UNIT    000006      DBDRV     DKDRV     DMDRV     DRDRV     DXDRV
                  #   EXEDF     IOSUB     MMDRV     TTDRV
U.VCB     000034      DREIF     DRQIO     DTDRV   # EXEDF     MMDRV
U2.CRT    002000    # EXEDF     TTDRV
U2.ESC    001000    # EXEDF     TTDRV
U2.HLD    000040    # EXEDF     TTDRV
U2.LOG    000400    # EXEDF     TTDRV
U2.LWC    000001    # EXEDF     TTDRV
V$$CTR    000400    # SYSDF
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

W.BATT    000006    # EXEDF     TTDRV
W.BLGH    000020    # EXEDF     TTDRV
W.BOFF    000012    # EXEDF     TTDRV
X.AST     000032    # SYSDF
X.DSI     000024    # SYSDF
X.FORK    000012    # SYSDF
X.ISR     000010    # SYSDF
X.JSR     000002    # SYSDF
X.LEN     000050    # SYSDF
X.LNK     000000    # SYSDF
X.PSW     000006    # SYSDF
X.REL     000022    # SYSDF
X.TCB     000026    # SYSDF
X.VEC     000044    # SYSDF
X.VPC     000046    # SYSDF
$ABCTK    016370-R    DRATX     DREIF   # REQSB     SSTSR     SYSXT
$ABTIM    006174-R    QUEUE   # SYSCM     TDSCH
$ABTSK    016374-R    DRABO     LOADR     PARTY   # REQSB
$ACHCK    007704-R    ADDRV     DRATX     DRQIO   # IOSUB     SSTSR
                      SYSXT
$ACHKB    007712-R    DRQIO   # IOSUB     TTDRV
$ACHKP    007650-R    DRDSP     DRGLI     DRGPP     DRGTK     DRGTP
                      DRMAP     DRRAS     DRSED   # IOSUB
$ACHKW    007674-R    DRSST   # IOSUB
$ACTHD    006172-R    DREIF     DRSED     POWER     REQSB   # SYSCM
                      TDSCH
$ACTRM    017102-R    DREIF     DRRES   # REQSB
$ACTTK    016562-R    DRRES     LOADR   # REQSB
$ADTBL    075350-R  # ADDRV
$ALCLK    007300-R  # CORAL     DRMKT     DRPUT
$ALEB1    034644-R  # ERROR
$ALEMB    034630-R  # ERROR     PARTY     POWER
$ALOCB    007166-R  # CORAL     DMDRV     DREIF     DRQIO     DRREG
                      ERROR     GRDRV     IOSUB     LOADR     PLSUB
                      REQSB     TTDRV
$ALOC1    007230-R  # CORAL
$ALPKT    007314-R  # CORAL     DRMAP     DRQIO     DRRAS
$ASUMR    013154-R  # IOSUB
$BILDS    016450-R    LOADR   # REQSB
$BLKCK    010640-R    DBDRV     DKDRV     DMDRV     DRDRV     DTDRV
                    # IOSUB
$BLKC1    010650-R    DMDRV     DRDRV   # IOSUB
$BLXIO    006650-R  # BFCTL
$BMSET    034740-R    DBDRV     DKDRV     DMDRV     DRDRV     DXDRV
                    # ERROR     MMDRV
$BTMSK    006264-R    DRQIO     IOSUB   # SYSCM     SYSTB     TTDRV
$BTSTP    002626-R  # CRASH     PANIC
$CEFI     010024-R    DRQIO     GRDRV   # IOSUB     REQSB
$CEFN     010020-R    DRDSP     DRMAP     DRQIO   # IOSUB
$CFLPT    006212-R    REQSB   # SYSCM
$CHKPT    020136-R    DREXP     LOADR   # REQSB
$CLKHD    006232-R    QUEUE   # SYSCM     TDSCH
$CLPAR    014666-R  # PARTY     POWER
$CLRMV    016174-R    DRCMT   # QUEUE
$CMBEG    006116-R  # SYSCM
$CMEND    006430-R  # SYSCM
$COMEF    006122-R    DRSED     IOSUB     REQSB   # SYSCM
$COPT     006234-R    REQSB   # SYSCM
$CRALT    001674-R  # CRASH     EXDBT
$CRASH    001664-R  # CRASH     LOWCR     SSTSR
$CRATT    036034-R    DRMAP     DRREG     LOADR   # PLSUB
$CRAVL    006166-R    CORAL     INITL   # SYSCM
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

$CRLF     052566-R    CRASH   # PANIC
$CRPAS    013634-R    DBDRV     DKDRV     DRDRV     DTDRV   # IOSUB
                      MMDRV
$CRPBF    001114-R  # CRASH
$CRPST    001160-R  # CRASH
$CRSBF    001124-R  # CRASH
$CRSBN    001656-R  # CRASH
$CRSCS    001662-R  # CRASH
$CRSHT    002404-R  # CRASH
$CRSST    001654-R  # CRASH
$CRSUN    002410-R  # CRASH
$CRUPC    000632-R  # CRASH     EXDBT
$CRUST    000634-R  # CRASH     EXDBT
$CURPR    006121-R    REQSB   # SYSCM     SYSXT
$CVRTM    007532-R  # CVRTM     DRMKT
$C5TA     006430-R  # C5TA      DREIF
$DASTT    016740-R    DREIF     DRMAP     DRRAS     LOADR     POWER
                    # REQSB     SSTSR
$DBINT    051340-R  # DBDRV
$DBTBL    050634-R  # DBDRV
$DB0      104756-R  # SYSTB
$DEACB    007334-R  # CORAL     DMDRV     DREIF     DRGCL     DRREG
                      INITL     LOADR     REQSB     SYSXT     TTDRV
$DEAC1    007374-R  # CORAL
$DECLK    007306-R  # CORAL     DRPUT     QUEUE     TDSCH
$DEPKT    007330-R  # CORAL     DREIF     DRMAP     DRQIO     DRRAS
                      IOSUB
$DETRG    033512-R    DREIF   # DRREG
$DEUMR    013272-R  # IOSUB
$DEVHD    006204-R    DRASG     INITL     IOSUB   # SYSCM
$DEVTB    104512-R    SYSCM   # SYSTB
$DHINP    045132-R  # TTDRV
$DHOUT    044600-R  # TTDRV
$DIRSV    002634-R    DRDSP     PARTY     SSTSR   # SYSXT
$DIRXT    003100-R    INITL   # SYSXT
$DIV      013730-R    C5TA      DBDRV     DKDRV     DMDRV     DRDRV
                      DRGLI     EXDBT     INITL   # IOSUB
$DKINT    062702-R  # DKDRV
$DKTBL    062272-R  # DKDRV
$DPLM1    023610-R  # DRDSP     EXDBT     SSTSR
$DPLM2    023614-R  # DRDSP     EXDBT     SSTSR
$DQLM1    030472-R  # DRQIO     EXDBT     SSTSR
$DQLM2    030502-R  # DRQIO     EXDBT     SSTSR
$DQUMR    013472-R  # IOSUB
$DRABO    024066-R  # DRABO     DRDSP
$DRASG    024112-R  # DRASG     DRDSP
$DRATP    034030-R    DRDSP   # DRRES
$DRATR    033176-R    DRDSP   # DRREG
$DRATX    024336-R  # DRATX     DRDSP
$DRCEF    034350-R    DRDSP   # DRSED
$DRCMT    024504-R  # DRCMT     DRDSP     DREIF
$DRCRR    032610-R    DRDSP   # DRREG
$DRCRW    026052-R    DRDSP   # DRMAP
$DRCSR    024510-R  # DRCMT     DRDSP
$DRDAR    024524-R  # DRDAR     DRDSP
$DRDCP    024560-R  # DRDCP     DRDSP
$DRDSE    034360-R    DRDSP     DREIF     DRMAP     DRRAS     DRRES
                    # DRSED     POWER     REQSB     TDSCH
$DRDTR    033360-R    DRDSP   # DRREG
$DREAR    024540-R  # DRDAR     DRDSP
$DRECP    024604-R  # DRDCP     DRDSP
$DREIF    004674-R    DRDSP   # DREIF
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

$DRELW    026300-R    DRDSP   # DRMAP
$DREXP    024630-R    DRDSP   # DREXP
$DREXT    004702-R    DRDSP   # DREIF     DRMAP     DRRAS     SYSXT
$DRFEX    030066-R    DRDSP   # DRPUT
$DRGCL    025354-R    DRDSP   # DRGCL
$DRGLI    025462-R    DRDSP   # DRGLI
$DRGMX    027472-R    DRDSP   # DRMAP
$DRGPP    025564-R    DRDSP   # DRGPP
$DRGSS    025666-R    DRDSP   # DRGSS
$DRGTK    025676-R    DRDSP   # DRGTK
$DRGTP    026016-R    DRDSP   # DRGTP
$DRINT    047202-R  # DRDRV
$DRLM1    023514-R  # DRDSP     EXDBT     SSTSR
$DRLM2    023534-R  # DRDSP     EXDBT     SSTSR
$DRMAP    026334-R    DRDSP   # DRMAP
$DRMKT    027716-R    DRDSP   # DRMKT
$DRPUT    030110-R    DRDSP   # DRPUT
$DRQIO    030376-R    DRDSP   # DRQIO
$DRQRQ    031472-R    DREIF   # DRQIO
$DRRAF    034374-R    DRDSP   # DRSED
$DRRCV    030150-R    DRDSP   # DRPUT
$DRREC    032262-R    DRDSP   # DRRAS
$DRREQ    033724-R    DRDSP   # DRREQ
$DRRES    033756-R    DRDSP   # DRRES
$DRSPN    034016-R    DRDSP   # DRRES
$DRSRF    026704-R    DRDSP   # DRMAP
$DRSTV    034550-R    DRDSP   # DRSST
$DRTBL    046372-R  # DRDRV
$DRUNM    026660-R    DRDSP   # DRMAP
$DRWFL    034460-R    DRDSP   # DRSED
$DRWFS    034522-R    DRDSP     DRQIO   # DRSED
$DRWSE    034444-R    DRDSP   # DRSED
$DR0      105470-R  # SYSTB
$DS0      105672-R  # SYSTB
$DTINT    071066-R  # DTDRV
$DTOER    034764-R    DBDRV     DKDRV     DMDRV     DRDRV     DTDRV
                      DXDRV   # ERROR     MMDRV
$DTTBL    070444-R  # DTDRV
$DT0      106214-R  # SYSTB
$DVCER    035026-R  # ERROR
$DVERR    035026-R    DBDRV     DKDRV     DMDRV     DRDRV     DTDRV
                      DXDRV   # ERROR     MMDRV
$DVMSG    010110-R    DTDRV   # IOSUB     LOADR     LPDRV     MMDRV
$DXINT    074122-R  # DXDRV
$DXTBL    073532-R  # DXDRV
$DX0      106424-R  # SYSTB
$DYPMN    006250-R  # SYSCM     TDSCH
$EDIT     052614-R    CRASH   # PANIC
$EMSST    021070-R    DRDSP   # SSTSR
$EMTRP    023440-R  # DRDSP     EXDBT     LOWCR
$ERRHD    006324-R    ERROR   # SYSCM
$ERRLM    006330-R    ERROR   # SYSCM
$ERRPT    006210-R    ERROR   # SYSCM
$ERRSQ    006332-R    DMDRV     ERROR   # SYSCM
$ERRSV    006334-R  # SYSCM
$ERRSZ    006336-R    ERROR   # SYSCM
$EXRQF    020344-R    IOSUB     QUEUE   # REQSB
$EXRQN    020362-R    DREIF   # REQSB     TDSCH     TTDRV
$EXRQP    020336-R    IOSUB   # REQSB
$EXSIZ    006150-R    CORAL   # SYSCM
$FINBF    004452-R    DREIF   # SYSXT
$FLTRP    021124-R    EXDBT   # SSTSR
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

$FMASK    006226-R  # SYSCM
$FNDSP    017660-R    DRREG   # REQSB
$FORK     002670-R    DBDRV     DKDRV     DMDRV     DRDRV     DTDRV
                      DXDRV     LPDRV     MMDRV   # SYSXT     TTDRV
$FORK0    002712-R    IOSUB     SSTSR   # SYSXT     TDSCH
$FORK1    002710-R    ERROR     GRDRV   # SYSXT
$FPPRQ    021146-R    INITL   # SSTSR
$FPPR7    021124-R    POWER   # SSTSR
$FPPR8    021132-R    INITL   # SSTSR
$FRKHD    006222-R  # SYSCM     SYSXT
$GRFRK    000000      GRDRV
$GTWRD    006616-R  # BFCTL
$HEADR    006116-R    DRATX     DRDSP     DREIF     DRRAS     IOSUB
                      LOADR     PARTY     POWER     SSTSR   # SYSCM
                      SYSXT     SYTAB
$ICHKP    020070-R    IOSUB   # REQSB
$ICINT    075320-R  # ICDRV
$ICTBL    074676-R  # ICDRV
$IDLCT    006244-R  # SYSCM     SYSXT
$IDLFL    006245-R  # SYSCM     SYSXT
$IDLPT    006246-R  # SYSCM     SYSXT
$ILINS    021232-R    EXDBT     LOWCR   # SSTSR
$INITL    117656-R    EXDBT   # INITL
$INTCT    006220-R  # SYSCM     TDSCH
$INTSE    002762-R  # SYSXT
$INTSV    003020-R    ERROR     GRDRV     SSTSR   # SYSXT     TDSCH
                      UDDRV
$INTXT    002760-R    NLDRV   # SYSXT     UDDRV
$INTX1    003042-R  # SYSXT
$IOABM    006340-R    DTDRV     ERROR   # SYSCM     SYSXT
$IOALT    010736-R  # IOSUB     TTDRV
$IODON    010740-R    DBDRV     DKDRV     DMDRV     DRDRV     DTDRV
                      DXDRV   # IOSUB     LPDRV     MMDRV     TTDRV
$IOFIN    011116-R    ADDRV     DRQIO     GRDRV     ICDRV   # IOSUB
                      ISDRV     NLDRV     SYSXT     TTDRV     UDDRV
$IOKIL    011374-R    DRASG     DREIF     DRQIO   # IOSUB
$IOTRP    021244-R    EXDBT     LOWCR   # SSTSR
$ISINT    075666-R  # ISDRV
$ISTBL    075570-R  # ISDRV
$LCKPR    012314-R  # IOSUB
$LDPWF    016034-R  # POWER
$LDRPT    006240-R    IOSUB     REQSB   # SYSCM
$LOADR    102600-R  # LOADR     SYTAB
$LOADT    020332-R  # REQSB
$LOGHD    006156-R    DRASG   # SYSCM
$LPINT    067622-R  # LPDRV
$LPTBL    067464-R  # LPDRV
$LP0      106546-R  # SYSTB
$LSTLK    006162-R    DREIF   # SYSCM
$MAPTK    020746-R    DREXP     LOADR   # REQSB
$MCRCB    006160-R    DRGCL   # SYSCM
$MCRPT    006206-R    QUEUE   # SYSCM
$MMINT    054762-R  # MMDRV
$MMTBL    053406-R  # MMDRV
$MM0      106746-R  # SYSTB
$MPCSR    014130-R  # PARTY     POWER
$MPCTL    014126-R  # PARTY     POWER
$MPLND    011634-R    DRASG     DRQIO   # IOSUB
$MPLNE    011614-R    DREIF     DRRES   # IOSUB
$MPLUN    011576-R    DRASG     DRGLI     DRQIO   # IOSUB
$MPPHY    011676-R    DRQIO     GRDRV   # IOSUB
$NL0      117070-R  # SYSTB
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

$NONSI    003142-R    INITL     LOWCR   # SYSXT
$NS0      035310-R  # ERROR     LOWCR
$NS1      035316-R  # ERROR     LOWCR
$NS2      035324-R  # ERROR     LOWCR
$NS3      035332-R  # ERROR     LOWCR
$NS4      035340-R  # ERROR
$NS5      035346-R  # ERROR
$NS6      035354-R  # ERROR
$NS7      035362-R  # ERROR
$NXTSK    017232-R    DRATX     DRDCP     DREXP     DRRES     IOSUB
                    # REQSB     TDSCH
$OUT      052664-R    CRASH   # PANIC
$OUTB     052660-R    CRASH   # PANIC
$PANIC    052562-R  # PANIC
$PARHD    006236-R    PLSUB   # SYSCM     TDSCH
$PARPT    006230-R  # SYSCM
$PARTB    014034-R    INITL   # PARTY     SYSCM
$PCBS     117426-R    SYSCM   # SYTAB
$PKAVL    006410-R    CORAL   # SYSCM
$PKMAX    006413-R    CORAL   # SYSCM
$PKNUM    006412-R  # SYSCM
$POOL     117552-R    CORAL   # INITL
$POWER    015664-R  # POWER     SYSXT
$PTBYT    006570-R  # BFCTL     TTDRV
$PTWRD    006616-R    ADDRV   # BFCTL
$PWRFL    006152-R    POWER   # SYSCM     SYSXT
$QASTT    016772-R  # REQSB     TDSCH     TTDRV
$QEMB     035526-R  # ERROR     IOSUB     PARTY     POWER
$QINSF    016242-R    DRMAP     DRRAS     DRRES     ERROR     GRDRV
                      IOSUB     PLSUB   # QUEUE     REQSB
$QINSP    016250-R    DRQIO     DRRES     LOADR     PLSUB   # QUEUE
                      REQSB     TTDRV
$QMCRL    016306-R    DREIF   # QUEUE     TTDRV
$QRMVF    016316-R    DREIF     DRMAP     DRRES     LOADR
                    # QUEUE     SYSXT
$QRMVT    016330-R    DRRAS     DRREG     DRRES   # QUEUE     REQSB
$RELOC    012764-R    ADDRV     DRQIO     GRDRV   # IOSUB     SYSXT
                      TTDRV
$RELOM    013034-R    DRATX     DRQIO   # IOSUB     SSTSR     SYSXT
$RELOP    013554-R    DBDRV     DMDRV     DRDRV   # IOSUB
$RLMCB    025416-R    DREIF   # DRGCL
$RLPAR    017156-R    DREIF     LOADR   # REQSB
$RLPR1    017220-R    DRREG   # REQSB
$RQSCH    006200-R    DRSED     POWER     REQSB   # SYSCM     SYSXT
$SAVNR    004620-R    ERROR     IOSUB     LOADR     REQSB   # SYSXT
$SCDVT    013050-R  # IOSUB     POWER     TDSCH
$SCDV1    013054-R  # IOSUB
$SETCR    016614-R    DRRES     GRDRV     IOSUB   # REQSB
$SETF     016674-R    DRRAS     IOSUB   # REQSB
$SRATT    036134-R    DRGPP     DRMAP     DRREG   # PLSUB
$SRNAM    035654-R    DRGPP     DRREG   # PLSUB
$SRSTD    017042-R    DRDSP   # REQSB
$SRWND    036172-R    DRMAP   # PLSUB
$STACK    000632-R    CRASH     DRDSP     INITL   # LOWCR     SSTSR
                      SYSXT
$STD      117462-R    SYSCM   # SYTAB
$STKDP    006202-R    DRDSP     EXDBT     SSTSR   # SYSCM     SYSXT
$STMAP    013316-R    DKDRV     DMDRV     DTDRV   # IOSUB
$STPCT    017132-R    LOADR   # REQSB
$STPTK    017136-R  # REQSB     TTDRV
$SWSTK    004640-R    DRDSP   # SYSXT
$SYBEG    120756-R  # INITL     SYSCM
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

$SYSID    006126-R    EXDBT     INITL   # SYSCM
$SYSIZ    006342-R    INITL   # SYSCM
$SYTOP    124756-R  # INITL     SYSCM
$SYUIC    006144-R  # SYSCM
$TKNPT    006132-R    DREIF     IOSUB     REQSB   # SYSCM
$TKPS     006370-R    CVRTM     DRGTP     INITL   # SYSCM     TDSCH
                      TTDRV
$TKTCB    006176-R    DRDSP     DREIF     DRQIO     DRSED     IOSUB
                      LOADR     PARTY     REQSB     SSTSR   # SYSCM
                      SYSXT
$TKWSE    034440-R    DREIF     DRQIO   # DRSED     LOADR
$TRACE    021324-R    EXDBT     LOWCR   # SSTSR
$TRP04    021336-R    EXDBT     LOWCR   # SSTSR
$TRTRP    023374-R  # DRDSP     EXDBT     LOWCR
$TSKHD    006242-R    REQSB   # SYSCM
$TSKRP    020414-R    DRREQ   # REQSB
$TSKRQ    020412-R  # REQSB
$TSKRT    020406-R    ERROR   # REQSB     TDSCH
$TSTCP    017762-R  # REQSB
$TTNS     006406-R    DRGTP     ERROR     PARTY   # SYSCM     TDSCH
$TTTBL    044570-R  # TTDRV
$TT0      107130-R  # SYSTB
$TT1      114556-R  # SYSTB
$TT10     115046-R  # SYSTB
$TT11     115076-R  # SYSTB
$TT12     115126-R  # SYSTB
$TT13     115156-R  # SYSTB
$TT14     115206-R  # SYSTB
$TT15     115236-R  # SYSTB
$TT16     115266-R  # SYSTB
$TT17     115316-R  # SYSTB
$TT2      114626-R  # SYSTB
$TT20     115346-R  # SYSTB
$TT21     115376-R  # SYSTB
$TT22     115426-R  # SYSTB
$TT23     115456-R  # SYSTB
$TT24     115506-R  # SYSTB
$TT25     115536-R  # SYSTB
$TT26     115566-R  # SYSTB
$TT27     115616-R  # SYSTB
$TT3      114656-R  # SYSTB
$TT30     115646-R  # SYSTB
$TT31     115676-R  # SYSTB
$TT32     115726-R  # SYSTB
$TT33     115756-R  # SYSTB
$TT34     116006-R  # SYSTB
$TT35     116036-R  # SYSTB
$TT36     116066-R  # SYSTB
$TT37     116116-R  # SYSTB
$TT4      114706-R  # SYSTB
$TT40     116146-R  # SYSTB
$TT41     116176-R  # SYSTB
$TT42     116226-R  # SYSTB
$TT43     116256-R  # SYSTB
$TT44     116306-R  # SYSTB
$TT45     116336-R  # SYSTB
$TT46     116366-R  # SYSTB
$TT47     116416-R  # SYSTB
$TT5      114736-R  # SYSTB
$TT50     116446-R  # SYSTB
$TT51     116476-R  # SYSTB
$TT52     116526-R  # SYSTB
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

$TT53     116556-R  # SYSTB
$TT54     116606-R  # SYSTB
$TT55     116636-R  # SYSTB
$TT56     116666-R  # SYSTB
$TT57     116716-R  # SYSTB
$TT6      114766-R  # SYSTB
$TT60     116746-R  # SYSTB
$TT7      115016-R  # SYSTB
$UDINT    050560-R  # UDDRV
$UDTBL    050472-R  # UDDRV
$UISET    020706-R    DRMKT     DRREQ   # REQSB
$UMRHD    006416-R    IOSUB   # SYSCM
$UMRWT    006424-R    IOSUB   # SYSCM
$UNMAP    036232-R    DRMAP     DRREG   # PLSUB
$USRTB    000000      SYSTB
$WTUMR    013516-R  # IOSUB
$XDT      076362-R  # EXDBT
.CL0      117220-R  # SYSTB
.CO0      117174-R    SYSCM   # SYSTB
.DB0      104552-R  # SYSTB
.DB1      104612-R  # SYSTB
.DB2      104652-R  # SYSTB
.DB3      104712-R  # SYSTB
.DK0      105054-R  # SYSTB
.DK1      105114-R  # SYSTB
.DK2      105154-R  # SYSTB
.DT0      105770-R  # SYSTB
.DT1      106034-R  # SYSTB
.DT2      106100-R  # SYSTB
.DT3      106144-R  # SYSTB
.DX0      106320-R  # SYSTB
.DX1      106360-R  # SYSTB
.LB0      117244-R    INITL   # SYSTB     SYTAB
.LDR      117462-R    SYSCM   # SYTAB
.LDRHD    117302-R  # SYTAB
.LP0      106514-R  # SYSTB
.MM0      106636-R  # SYSTB
.MM1      106700-R  # SYSTB
.NL0      117036-R  # SYSTB
.SY0      117270-R    INITL   # SYSTB     SYTAB
.TI0      117150-R  # SYSTB
.TT0      107046-R  # SYSTB
.TT1      107222-R  # SYSTB
.TT10     110050-R  # SYSTB
.TT11     110142-R  # SYSTB
.TT12     110234-R  # SYSTB
.TT13     110326-R  # SYSTB
.TT14     110420-R  # SYSTB
.TT15     110512-R  # SYSTB
.TT16     110604-R  # SYSTB
.TT17     110676-R  # SYSTB
.TT2      107314-R  # SYSTB
.TT20     110770-R  # SYSTB
.TT21     111062-R  # SYSTB
.TT22     111154-R  # SYSTB
.TT23     111246-R  # SYSTB
.TT24     111340-R  # SYSTB
.TT25     111432-R  # SYSTB
.TT26     111524-R  # SYSTB
.TT27     111616-R  # SYSTB
.TT3      107406-R  # SYSTB
.TT30     111710-R  # SYSTB
```

```
Symbol  Value       Modules That Reference Symbol

.TT31     112002-R  # SYSTB
.TT32     112074-R  # SYSTB
.TT33     112166-R  # SYSTB
.TT34     112260-R  # SYSTB
.TT35     112352-R  # SYSTB
.TT36     112444-R  # SYSTB
.TT37     112536-R  # SYSTB
.TT4      107500-R  # SYSTB
.TT40     112630-R  # SYSTB
.TT41     112722-R  # SYSTB
.TT5      107572-R  # SYSTB
.TT50     113550-R  # SYSTB
.TT51     113642-R  # SYSTB
.TT52     113734-R  # SYSTB
.TT53     114026-R  # SYSTB
.TT54     114120-R  # SYSTB
.TT55     114212-R  # SYSTB
.TT56     114304-R  # SYSTB
.TT57     114376-R  # SYSTB
.TT6      107664-R  # SYSTB
.TT60     114470-R  # SYSTB
.TT7      107756-R  # SYSTB
```

9.3 MCRMU GLOBAL CROSS-REFERENCE

This cross-reference is for a mapped system.

The cross-reference contains an alphabetic listing of each global symbol along with its value and the name of each referencing module. When a symbol is defined in several segments within an overlay structure, TKB prints the last defined value in the listing. Similarly, in a real TKB cross-reference listing, TKB would print the module name more than once for each symbol if the module is loaded in several segments within the structure.

The Task Builder creates an MCRMU.CRF cross-reference file when /CR is specified in the Task Builder command file used to build MCRMU. One of the input files to the Task Builder when building MCRMU is the Executive symbol table file, RSX11M.STB. RSX11M.STB is needed because MCRMU references some Executive symbols. All the symbols from RSX11M.STB are put in the MCRMU.CRF symbol table file even though they are not referenced by MCR. Therefore, some symbols appearing here in the MCRMU cross-reference are defined in the Executive but not used by MCRMU. These symbols are shown defined in the Executive LOWCR or EXEDF modules.

The value contains the suffix -R if the symbol is relocatable.

CROSS-REFERENCES

Prefix symbols accompanying each module name define the type of reference as follows:

Prefix Symbol — Reference Type

blank — Module contains a reference that is resolved in the same segment or in a segment toward the root.

^ — Module contains a reference that is resolved directly in a segment away from the root or in a co-tree.

@ — Module contains a reference that is resolved through an autoload vector.

— Module contains a non-autoloadable definition. This module defines the symbol.

* — Module contains an autoloadable definition. This module defines the symbol.

```
Symbol  Value      Modules That Reference Symbol

C.SCHD   000002    # EXEDF     FIXOV
DV.PSE   010000    # EXEDF   # LOWCR
DV.TTY   000004    # EXEDF   # LOWCR
DV.UMD   000200    # EXEDF   # LOWCR
D$$YNM   000000    # LOWCR
D.DSP    000012    # EXEDF   # LOWCR
D.LNK    000000    # EXEDF     FIXOV
D.MSK    000014    # EXEDF   # LOWCR
D.NAM    000004    # EXEDF     FIXOV     FMTDV     GTTSK    # LOWCR
                     MCRDIS
D.PCB    000034    # EXEDF   # LOWCR
D.RS00   000000    # LOWCR
D.RS1    177777    # LOWCR
D.RS10   177766    # LOWCR
D.RS16   177760    # LOWCR
D.RS17   177757    # LOWCR
D.RS19   177755    # LOWCR
D.RS2    177776    # LOWCR
D.RS22   000002    # LOWCR
D.RS5    177773    # LOWCR
D.RS6    177772    # LOWCR
D.RS7    177771    # LOWCR
D.RS8    177770    # LOWCR
D.RS80   177660    # LOWCR
D.RS81   177657    # LOWCR
D.RS84   177654    # LOWCR
D.RS85   177653    # LOWCR
D.RS86   177652    # LOWCR
D.RS87   177651    # LOWCR
D.RS90   177646    # LOWCR
D.RS92   177644    # LOWCR
D.RS93   177643    # LOWCR
D.RS94   177642    # LOWCR
D.RS95   177641    # LOWCR
D.RS96   177640    # LOWCR
D.RS97   177637    # LOWCR
D.RS98   177636    # LOWCR
D.RS99   177635    # LOWCR
D.UCB    000002    # EXEDF     FIXOV     FMTDV     GTMNM    # LOWCR
```

```
Symbol  Value       Modules That Reference Symbol

D.UCBL    000010    # EXEDF     FIXOV     FMTDV     GTMNM   # LOWCR
D.UNIT    000006    # EXEDF     FIXOV     FMTDV     GTMNM   # LOWCR
D.VCAN    000002    # EXEDF   # LOWCR
D.VINI    000000    # EXEDF   # LOWCR
D.VOUT    000004    # EXEDF   # LOWCR
D.VPWF    000006    # EXEDF   # LOWCR
EC.DTO    000140    # EXEDF   # LOWCR
EC.DVC    000001    # EXEDF   # LOWCR
EC.NSI    000141    # EXEDF   # LOWCR
E.LGTH    000056    # EXEDF   # LOWCR
E.OPC     000022    # EXEDF   # LOWCR
E.RTRY    000016    # EXEDF   # LOWCR
E.SIZE    000000    # EXEDF   # LOWCR
FE.CAL    000040    # EXEDF     MCRDIS
FE.MUP    000002    # EXEDF     FIXOV     MCRDIS
FE.MXT    040000    # EXEDF     MCRDIS
FE.PLA    000020    # EXEDF     FIXOV     MCRDIS
H.HDLN    000002    # EXEDF     FIXOV
H.LUN     000076    # EXEDF     FIXOV
H.WND     000044    # EXEDF     MCROOT
IE.ABO    177761    # LOWCR
IE.ALN    177736    # LOWCR
IE.BAD    177777    # LOWCR
IE.BLK    177754    # LOWCR
IE.BYT    177755    # LOWCR
IE.DAA    177770    # LOWCR
IE.DNA    177771    # LOWCR
IE.DNR    177775    # LOWCR
IE.EOF    177766      MCRDIS
IE.IFC    177776    # LOWCR
IE.LCK    177745    # LOWCR
IE.NLN    177733    # LOWCR
IE.NOD    177751    # LOWCR
IE.OFL    177677    # LOWCR
IE.OVR    177756    # LOWCR
IE.PRI    177760    # LOWCR
IE.SPC    177772    # LOWCR
IE.ULK    177653    # LOWCR
IO.ATT    001400      LN1OV   # LOWCR
IO.CLN    003400    # LOWCR
IO.DET    002000    # LOWCR     LUNOV
IO.KIL    000012      ERROV     MCRDIS
IO.LOV    001010    # LOWCR
IO.NLK    011400      FIXOV
IO.RLB    001000      LN1OV   # LOWCR
IO.RVB    010400    # LOWCR
IO.ULK    005000    # LOWCR
IO.WLB    000400    # LOWCR
IO.WVB    011000      ERROV     LN1OV   # LOWCR     MCRDIS
IQ.UMD    000004    # LOWCR
IS.SUC    000001    # LOWCR
KISAR5    172352    # EXEDF   # LOWCR
KISAR6    172354    # EXEDF   # LOWCR
L.ASG     000010    # EXEDF   # LOWCR
L.NAM     000002    # EXEDF   # LOWCR
L.TYPE    000005    # EXEDF   # LOWCR
L.UCB     000006    # EXEDF   # LOWCR
L.UNIT    000004    # EXEDF   # LOWCR
M$$MGE    000000    # LOWCR
P.ATT     000036      FIXOV   # LOWCR
P.BLKS    000016    # EXEDF   # LOWCR
P.BUSY    000024    # EXEDF   # LOWCR
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

P.HDR     000032      FIXOV   # LOWCR
P.IOC     000003    # EXEDF   # LOWCR
P.LGTH    000042      FIXOV   # LOWCR
P.LNK     000000    # EXEDF   # LOWCR
P.MAIN    000012    # EXEDF   # LOWCR
P.NAM     000004    # EXEDF   # LOWCR
P.OWN     000026    # EXEDF   # LOWCR
P.PRI     000002    # EXEDF   # LOWCR
P.PRO     000034    # LOWCR
P.REL     000014    # EXEDF   # LOWCR
P.SIZE    000016    # EXEDF   # LOWCR
P.STAT    000030    # EXEDF   # LOWCR
P.SUB     000010    # EXEDF   # LOWCR
P.SWSZ    000022    # EXEDF   # LOWCR
P.TCB     000026    # EXEDF   # LOWCR
P.WAIT    000020    # EXEDF   # LOWCR
SP.EIP    000001    # EXEDF   # LOWCR
SP.ENB    000002    # EXEDF   # LOWCR
S.BMSK    177776    # EXEDF   # LOWCR
S.BMSV    177774    # EXEDF   # LOWCR
S.CCB     000030    # LOWCR
S.CON     000010    # EXEDF   # LOWCR
S.CSR     000012    # EXEDF   # LOWCR
S.CTM     000006    # EXEDF   # LOWCR
S.DZCK    000030    # LOWCR
S.FRK     000016    # EXEDF   # LOWCR
S.ITM     000007    # EXEDF   # LOWCR
S.LHD     000000    # EXEDF   # LOWCR
S.MPR     000030    # LOWCR
S.PKT     000014    # EXEDF   # LOWCR
S.PRI     000004    # EXEDF   # LOWCR
S.RCNT    177772    # EXEDF   # LOWCR
S.ROFF    177773    # EXEDF   # LOWCR
S.STS     000011    # EXEDF   # LOWCR
S.VCT     000005    # EXEDF   # LOWCR
TS.EXE    100000    # EXEDF     FIXOV     MCRDIS
TS.OUT    000400    # EXEDF     FIXOV     MCRDIS
T.ACTL    000052    # EXEDF     MCRDIS
T.ATT     000054    # EXEDF     FIXOV
T.CPCB    000004    # EXEDF     MCRDIS
T.DPRI    000040    # EXEDF     MCRDIS
T.EXT     000000    # LOWCR
T.LGTH    000070      FIXOV   # LOWCR     MCRDIS
T.MXSZ    000050    # EXEDF     FIXOV
T.NAM     000006      ABOOV   # EXEDF     FIXOV
T.OFF     000060    # EXEDF     MCRDIS
T.PCB     000046    # EXEDF     FIXOV     LKLST     MCRDIS
T.PRI     000002    # EXEDF     MCRDIS
T.RCVL    000012    # EXEDF     FIXOV     MCRDIS
T.RRFL    000064    # EXEDF     FIXOV
T.STAT    000032      ABOOV   # EXEDF     FIXOV     MCRDIS
T.ST2     000034      ABOOV   # EXEDF     FIXOV     LKLST     MCRDIS
T.ST3     000036      ABOOV   # EXEDF     FIXOV     MCRDIS
T.TCBL    000030    # EXEDF     FIXOV     MCRDIS
T.UCB     000026      ABOOV   # EXEDF     GTMNM     GTTSK     MCRDIS
T2.ABO    000100      ABOOV   # EXEDF
T2.BFX    004000    # EXEDF     FIXOV
T2.CHK    020000    # EXEDF     FIXOV     MCRDIS
T2.CKD    010000    # EXEDF     LKLST
T2.FXD    002000    # EXEDF     FIXOV
T3.ACP    100000    # EXEDF     MCRDIS
T3.MCR    004000    # EXEDF     MCRDIS
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

T3.PMD    040000    # EXEDF     MCRDIS
T3.PRV    010000      ABOOV   # EXEDF     MCRDIS
T3.REM    020000    # EXEDF     FIXOV     MCRDIS
T3.RST    000400    # EXEDF     MCRDIS
T3.SLV    002000      ABOOV   # EXEDF
UC.ATT    000010    # EXEDF   # LOWCR
UC.KIL    000004    # EXEDF   # LOWCR
UC.LGH    000003    # EXEDF   # LOWCR
UC.NPR    000100    # EXEDF   # LOWCR
UC.PWF    000020    # EXEDF   # LOWCR
UC.QUE    000040    # EXEDF   # LOWCR
UISAR0    177640    # EXEDF   # LOWCR
UISDR0    177600    # EXEDF   # LOWCR
US.BSY    000200    # EXEDF   # LOWCR
US.FOR    000040    # EXEDF   # LOWCR
US.MDM    000020    # EXEDF   # LOWCR
US.MNT    000100    # EXEDF   # LOWCR
US.OFL    000001    # EXEDF   # LOWCR
US.PUB    000004    # EXEDF   # LOWCR
US.UMD    000010    # EXEDF   # LOWCR
U.ACP     000032    # EXEDF   # LOWCR
U.ATT     000022    # EXEDF   # LOWCR
U.BUF     000024    # EXEDF   # LOWCR
U.CNT     000030    # EXEDF   # LOWCR
U.CTL     000004    # EXEDF   # LOWCR
U.CW1     000010    # EXEDF   # LOWCR
U.CW2     000012      ABOOV   # EXEDF     FIXOV   # LOWCR     MCRDIS
U.CW3     000014    # EXEDF   # LOWCR
U.CW4     000016    # EXEDF     MCRDIS
U.DCB     000000    # EXEDF     FMTDV     GTMNM     GTTSK     MCRDIS
U.LUIC    177774    # EXEDF     MCRDIS
U.OWN     177776    # EXEDF   # LOWCR
U.RED     000002      ABOOV   # EXEDF     GTMNM   # LOWCR     MCRDIS
U.SCB     000020    # EXEDF   # LOWCR
U.STS     000005    # EXEDF   # LOWCR
U.ST2     000007    # EXEDF   # LOWCR
U.UIC     000052    # EXEDF     MCRDIS
U.UNIT    000006    # EXEDF   # LOWCR
U.VCB     000034    # EXEDF   # LOWCR
U2.AT.    000020    # EXEDF     MCRDIS
U2.HLD    000040    # EXEDF     MCRDIS
U2.LOG    000400    # EXEDF     FIXOV     MCRDIS
U2.PRV    000010      ABOOV   # EXEDF     FIXOV     MCRDIS
V$$CTR    000410    # LOWCR
W.BLVR    000002    # EXEDF     MCROOT
X.AST     000032    # LOWCR
X.DSI     000024    # LOWCR
X.FORK    000012    # LOWCR
X.ISR     000010    # LOWCR
X.JSR     000002    # LOWCR
X.LEN     000050    # LOWCR
X.LNK     000000    # LOWCR
X.PSW     000006    # LOWCR
X.REL     000022    # LOWCR
X.TCB     000026    # LOWCR
X.VEC     000044    # LOWCR
X.VPC     000046    # LOWCR
$ABCTK    014460    # LOWCR
$ABOEP    122036-R  # ABOOV     PR1OV
$ABTIM    005414    # LOWCR
$ABTSK    014464      ABOOV   # LOWCR
$ACHCK    007242    # LOWCR
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

$ACHKB    007250    # LOWCR
$ACHKP    007206    # LOWCR
$ACHKW    007232    # LOWCR
$ACTHD    005634    # LOWCR
$ACTRM    015172    # LOWCR
$ACTTK    014652    # LOWCR
$ALCLK    006636    # LOWCR
$ALEB1    032634    # LOWCR
$ALEMB    032620    # LOWCR
$ALOCB    006524      FIXOV   # LOWCR     MCRDIS
$ALOC1    006566    # LOWCR
$ALPKT    006652    # LOWCR
$BILDS    014540    # LOWCR
$BLKCK    010174    # LOWCR
$BLKC1    010204    # LOWCR
$BLXIO    006212    # LOWCR
$BMSET    032730    # LOWCR
$BTMSK    005640    # LOWCR
$CANEP    122122-R  # ABOOV     PR1OV
$CAT5     125444      GTTSK     MCRDIS    PR1OV
$CBDMG    123250      LUNOV
$CBOMG    123264      FMTDV     GTMNM
$CEFI     007362    # LOWCR
$CEFN     007356    # LOWCR
$CFLPT    005522    # LOWCR
$CHKPT    016226    # LOWCR
$CKACC    033726    # LOWCR
$CKCNT    005604    # LOWCR
$CKCSR    005606    # LOWCR
$CKINT    017726    # LOWCR
$CKLDC    005610    # LOWCR
$CLINS    014162    # LOWCR
$CLKHD    005556    # LOWCR
$CLRMV    014264      FIXOV   # LOWCR
$COMEF    005570    # LOWCR
$COPT     005560    # LOWCR
$CRASH    001470    # LOWCR
$CRATT    034024    # LOWCR
$CRAVL    005532    # LOWCR
$CRPAS    012470    # LOWCR
$CRSBF    000730    # LOWCR
$CRSBN    001462    # LOWCR
$CRSCS    001466    # LOWCR
$CRSHT    001752    # LOWCR
$CRSST    001460    # LOWCR
$CRSUN    001756    # LOWCR
$CVRTM    007070    # LOWCR
$C5TA     005772      ERROV   # LOWCR
$DASTT    015030    # LOWCR
$DB0      043366    # LOWCR
$DEACB    006672      FIXOV   # LOWCR     MCRDIS
$DEAC1    006732    # LOWCR
$DECLK    006644    # LOWCR
$DEPKT    006666      FIXOV   # LOWCR
$DETRG    031502      FIXOV   # LOWCR
$DEVHD    005462      FIXOV   # LOWCR
$DEVTB    043122    # LOWCR
$DIRSV    002264    # LOWCR
$DIRXT    002514    # LOWCR
$DIV      012564      FMTDV     GTMNM   # LOWCR
$DK0      043630    # LOWCR
$DPLM1    021662    # LOWCR
```

```
Symbol  Value       Modules That Reference Symbol

$DPLM2    021666    # LOWCR
$DQLM1    026526    # LOWCR
$DQLM2    026536    # LOWCR
$DRABO    022140    # LOWCR
$DRASG    022164    # LOWCR
$DRATP    032020      ABOOV     # LOWCR
$DRATR    031166    # LOWCR
$DRATX    022410    # LOWCR
$DRCEF    032340    # LOWCR
$DRCMT    022556    # LOWCR
$DRCRR    030600    # LOWCR
$DRCRW    024064    # LOWCR
$DRCSR    022562    # LOWCR
$DRDAR    022576    # LOWCR
$DRDCP    022632      EDCKP     # LOWCR
$DRDSE    032350      LKLST     # LOWCR
$DRDTR    031350    # LOWCR
$DREAR    022612    # LOWCR
$DRECP    022656      EDCKP     # LOWCR
$DREIF    004220    # LOWCR
$DRELW    024312    # LOWCR
$DREXP    022702    # LOWCR
$DREXT    004226    # LOWCR       MCRDIS
$DRFEX    026100    # LOWCR
$DRGCL    023372    # LOWCR
$DRGLI    023500    # LOWCR
$DRGMX    025504    # LOWCR
$DRGPP    023602    # LOWCR
$DRGSS    023704    # LOWCR
$DRGTK    023714    # LOWCR
$DRGTP    024030    # LOWCR
$DRLM1    021566    # LOWCR
$DRLM2    021606    # LOWCR
$DRMAP    024346    # LOWCR
$DRMKT    025730    # LOWCR
$DRPUT    026122    # LOWCR
$DRQIO    026410    # LOWCR
$DRQRQ    027516    # LOWCR
$DRRAF    032364    # LOWCR
$DRRCV    026162    # LOWCR
$DRREC    030306    # LOWCR
$DRREQ    031714    # LOWCR
$DRRES    031746    # LOWCR
$DRRRA    026142    # LOWCR
$DRRRF    025232    # LOWCR
$DRRUN    025762    # LOWCR
$DRSDV    032532    # LOWCR
$DRSEF    032420    # LOWCR
$DRSND    030440    # LOWCR
$DRSPN    032006    # LOWCR
$DRSRF    024716    # LOWCR
$DRSTV    032540    # LOWCR
$DRUNM    024672    # LOWCR
$DRWFL    032450    # LOWCR
$DRWFS    032512    # LOWCR
$DRWSE    032434    # LOWCR
$DSW      000046      ABOOV
$DS0      044064    # LOWCR
$DTOER    032754    # LOWCR
$DT0      044276    # LOWCR
$DVCER    033016    # LOWCR
$DVERR    033016    # LOWCR
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

$DVMSG    007446    # LOWCR
$DX0      044472    # LOWCR
$DYPMN    005470    # LOWCR
$EMSST    017156    # LOWCR
$EMTRP    021512    # LOWCR
$ERREP    121660-R  # ERROV
$ERRHD    005700    # LOWCR
$ERRLM    005704    # LOWCR
$ERRLN    000404    # ERRMSG    ERROV
$ERRPT    005514      FIXOV   # LOWCR
$ERRSQ    005706    # LOWCR
$ERRSV    005710    # LOWCR
$ERRSZ    005712    # LOWCR
$ERRTB    124064-R  # ERRMSG    ERROV
$EXRQF    016434    # LOWCR
$EXRQN    016452    # LOWCR
$EXRQP    016426    # LOWCR
$EXSIZ    005520    # LOWCR
$FINBF    003776    # LOWCR
$FIXEP    122714-R  # FIXOV     PR1OV
$FLTRP    017212    # LOWCR
$FMASK    005552      FIXOV   # LOWCR     MCRDIS
$FMTDV    123156-R  # FMTDV     LUNOV
$FNDSP    015750    # LOWCR
$FORK     002320    # LOWCR
$FORK0    002342    # LOWCR
$FORK1    002340    # LOWCR
$FPINT    017226    # LOWCR
$FRKHD    005546    # LOWCR
$GNBLK    125270-R  # GNBLK
$GTBYT    006102    # LOWCR
$GTMNM    125340-R  # GTMNM     GTTSK     MCRDIS
$GTPKT    007510    # LOWCR
$GTTSK    124714-R    ABOOV     FIXOV   # GTTSK     LN1OV
$GTWRD    006160    # LOWCR
$HEADR    005564      FIXOV     GTTSK     LKLST     LN1OV   # LOWCR
                      MCROOT
$ICHKP    016160    # LOWCR
$ILINS    017304    # LOWCR
$INITL    052414    # LOWCR
$INTCT    005542    # LOWCR
$INTSE    002376    # LOWCR
$INTSV    002434    # LOWCR
$INTXT    002374    # LOWCR
$INTX1    002456    # LOWCR
$IOABM    005714    # LOWCR
$IOALT    010272    # LOWCR
$IODON    010274    # LOWCR
$IOFIN    010414    # LOWCR
$IOKIL    010666    # LOWCR
$IOTRP    017316    # LOWCR
$LCKPR    011552    # LOWCR
$LDPWF    014124    # LOWCR
$LDRPT    005464    # LOWCR
$LN1EP    124436-R  # LN1OV     PR1OV
$LOAD     121004      MCROOT
$LOADR    041214    # LOWCR
$LOADT    016422    # LOWCR
$LOCKL    125120-R    FIXOV   # LKLST     MCRDIS
$LOGHD    005624    # LOWCR
$LP0      044614    # LOWCR
$LSTLK    005630      LKLST   # LOWCR
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

$LUNEP    122636-R  # LUNOV
$MAPTK    017036    # LOWCR
$MBUF     120362-R    ERROV     LN1OV     LUNOV     MCRDIS  # MCROOT
$MCKD     122212-R  # EDCKP     ERROV     MCRDIS
$MCKE     122224-R  # EDCKP     ERROV     MCRDIS
$MCMD     120454-R    ERROV     FIXOV     LN1OV     MCRDIS  # MCROOT
                      PR1OV
$MCOV     120464-R    MCRDIS  # MCROOT    PR1OV
$MCR      120634-R    ABOOV     ERROV     FIXOV     LUNOV   # MCROOT
$MCRCB    005626    # LOWCR     MCRDIS
$MCRPT    005466      FIXOV   # LOWCR
$MDIS     120500-R  # MCROOT
$MDPB     120366-R    ABOOV     ERROV     FIXOV     LN1OV     LUNOV
                      MCRDIS  # MCROOT
$MERLD    120720-R    ABOOV     FIXOV     LN1OV     LUNOV     MCRDIS
                    # MCROOT    PR1OV
$MERR     120470-R  # MCROOT
$MERRN    120474-R    ABOOV     ERROV     FIXOV     LN1OV     LUNOV
                      MCRDIS  # MCROOT    PR1OV
$MLDOV    120724-R    LN1OV     MCRDIS  # MCROOT
$MLIMI    120632-R  # MCROOT
$MLINE    120504-R    MCRDIS  # MCROOT
$MM0      045014    # LOWCR
$MOVRB    120416-R  # MCROOT
$MPARS    120424-R    ABOOV     ERROV     FIXOV     LN1OV     LUNOV
                      MCRDIS  # MCROOT    PR1OV
$MPCSR    012734    # LOWCR
$MPLND    011126    # LOWCR
$MPLNE    011106    # LOWCR
$MPLUN    011070    # LOWCR
$MPPHY    011170    # LOWCR
$MPPKT    011240    # LOWCR
$MPRSR    120460-R    MCRDIS  # MCROOT
$MPVBN    011372    # LOWCR
$MROOT    120706-R  # MCROOT
$MTERM    120476-R    MCRDIS  # MCROOT
$MUCB     120364-R    ABOOV     FIXOV     LN1OV     MCRDIS  # MCROOT
$MUL      012534    # LOWCR
$MXEXT    005770    # LOWCR
$NL0      051622    # LOWCR
$NNBLK    125272-R  # GNBLK     PR1OV
$NONSI    002556    # LOWCR
$NS0      033300    # LOWCR
$NS1      033306    # LOWCR
$NS2      033314    # LOWCR
$NS3      033322    # LOWCR
$NS4      033330    # LOWCR
$NS5      033336    # LOWCR
$NS6      033344    # LOWCR
$NS7      033352    # LOWCR
$NULL     043100    # LOWCR
$NXTSK    015322      FIXOV   # LOWCR
$OVEP     121634-R  # MCRDIS  ^ MCROOT
$PANIC    001470    # LOWCR
$PARHD    005422    # LOWCR
$PARPT    005554    # LOWCR
$PARTB    012670    # LOWCR
$PCBS     052164    # LOWCR
$PKAVL    005764    # LOWCR
$PKMAX    005767    # LOWCR
$PKNUM    005766    # LOWCR
$POOL     052310    # LOWCR
```

```
                           CROSS-REFERENCES

Symbol  Value       Modules That Reference Symbol

$POWER    013750    # LOWCR
$PP0      045144    # LOWCR
$PR0      045266    # LOWCR
$PR1EP    121654-R  # PR1OV
$PTBYT    006132    # LOWCR
$PTWRD    006160    # LOWCR
$PWRFL    005416    # LOWCR
$QASTT    015062    # LOWCR
$QEMB     033516    # LOWCR
$QINSF    014332    # LOWCR
$QINSP    014340      FIXOV   # LOWCR
$QMCRL    014376    # LOWCR
$QRMVF    014406      FIXOV   # LOWCR     MCRDIS
$QRMVT    014420    # LOWCR
$RELOC    012222    # LOWCR
$RELOM    012272    # LOWCR
$RELOP    012410    # LOWCR
$REMEP    122714-R  # FIXOV     PR1OV
$RESEP    122134-R  # ABOOV     PR1OV
$RLMCB    023434    # LOWCR
$RLPAR    015246      FIXOV   # LOWCR
$RLPR1    015310    # LOWCR
$RQSCH    005452    # LOWCR
$SAVNR    004144    # LOWCR
$SCDVT    012306    # LOWCR
$SCDV1    012312    # LOWCR
$SETCR    014704    # LOWCR
$SETF     014762    # LOWCR
$SETM     014766    # LOWCR
$SETRQ    014734    # LOWCR
$SETRT    014732    # LOWCR
$SGFLT    017336    # LOWCR
$SHFPT    005516      FIXOV   # LOWCR
$SIGFL    005420      LKLST   # LOWCR
$SRATT    034124    # LOWCR
$SRNAM    033644    # LOWCR
$SRSTD    015132      GTTSK   # LOWCR     MCRDIS
$SRWND    034162    # LOWCR
$STACK    000642    # LOWCR
$STD      052220    # LOWCR
$STKDP    005454    # LOWCR
$STPCT    015222    # LOWCR     MCRDIS
$STPTK    015226    # LOWCR
$SWSTK    004164    # LOWCR
$SYBEG    053424    # LOWCR
$SYSID    005574    # LOWCR
$SYSIZ    005716    # LOWCR
$SYTOP    063424    # LOWCR
$SYUIC    005612    # LOWCR
$TKNPT    005600      FIXOV   # LOWCR
$TKPS     005744    # LOWCR
$TKTCB    005446      ABOOV     EDCKP     GTMNM     GTTSK     LKLST
                    # LOWCR     MCRDIS
$TKWSE    032430      LKLST   # LOWCR
$TRACE    017376    # LOWCR
$TRP04    017410    # LOWCR
$TRTRP    021446    # LOWCR
$TSKHD    005512      FIXOV   # LOWCR
$TSKRP    016504    # LOWCR     MCRDIS
$TSKRQ    016502    # LOWCR
$TSKRT    016476    # LOWCR
$TSTCP    016052    # LOWCR
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

$TTNS     005762    # LOWCR
$TT0      045534    # LOWCR
$TT1      045564    # LOWCR
$TT10     047672    # LOWCR
$TT11     047722    # LOWCR
$TT12     047752    # LOWCR
$TT13     050002    # LOWCR
$TT14     050032    # LOWCR
$TT15     050062    # LOWCR
$TT16     050112    # LOWCR
$TT17     050142    # LOWCR
$TT2      047452    # LOWCR
$TT20     050172    # LOWCR
$TT21     050222    # LOWCR
$TT22     051210    # LOWCR
$TT23     051260    # LOWCR
$TT24     051310    # LOWCR
$TT25     051340    # LOWCR
$TT26     051370    # LOWCR
$TT27     051420    # LOWCR
$TT3      047502    # LOWCR
$TT30     051450    # LOWCR
$TT31     051500    # LOWCR
$TT4      047532    # LOWCR
$TT5      047562    # LOWCR
$TT6      047612    # LOWCR
$TT7      047642    # LOWCR
$UISET    016776    # LOWCR
$UNFEP    122714-R  # FIXOV     PR1OV
$UNLKL    125210-R    FIXOV   # LKLST     MCRDIS
$UNMAP    034222    # LOWCR
$USRTB    000000    # LOWCR
$XDT      035002    # LOWCR
.CL0      051756    # LOWCR
.CO0      051732    # LOWCR
.DB0      043162    # LOWCR
.DB1      043222    # LOWCR
.DB2      043262    # LOWCR
.DB3      043322    # LOWCR
.DK0      043464    # LOWCR
.DK1      043524    # LOWCR
.DK2      043564    # LOWCR
.DSW      000044    # LOWCR
.DS0      043720    # LOWCR
.DS1      043760    # LOWCR
.DS2      044020    # LOWCR
.DT0      044162    # LOWCR
.DT1      044226    # LOWCR
.DX0      044366    # LOWCR
.DX1      044426    # LOWCR
.LB0      052002    # LOWCR
.LDR      052220    # LOWCR
.LDRHD    052040    # LOWCR
.LP0      044562    # LOWCR
.MM0      044704    # LOWCR
.MM1      044746    # LOWCR
.NL0      051570    # LOWCR
.PP0      045112    # LOWCR
.PR0      045234    # LOWCR
.SY0      052026    # LOWCR
.TI0      051706    # LOWCR
```

```
                         CROSS-REFERENCES

Symbol  Value       Modules That Reference Symbol

.TT0     045362     # LOWCR
.TT1     045452     # LOWCR
.TT10    046400     # LOWCR
.TT11    046470     # LOWCR
.TT12    046560     # LOWCR
.TT13    046650     # LOWCR
.TT14    046740     # LOWCR
.TT15    047030     # LOWCR
.TT16    047120     # LOWCR
.TT17    047210     # LOWCR
.TT2     045660     # LOWCR
.TT20    047300     # LOWCR
.TT21    047370     # LOWCR
.TT22    050316     # LOWCR
.TT23    050406     # LOWCR
.TT24    050476     # LOWCR
.TT25    050566     # LOWCR
.TT26    050656     # LOWCR
.TT27    050746     # LOWCR
.TT3     045750     # LOWCR
.TT30    051036     # LOWCR
.TT31    051126     # LOWCR
.TT4     046040     # LOWCR
.TT5     046130     # LOWCR
.TT6     046220     # LOWCR
.TT7     046310     # LOWCR
```

9.4 MCRMU SEGMENT CROSS-REFERENCE

The MCRMU segment cross-reference lists the name of each overlay segment and the modules that compose it. The cross-reference follows:

```
Segment
Name             Resident Modules

ERROV              EDCKP   ERRMSG  ERROV
LUNOV              EXEDF   FMTDV   LUNOV
MCROOT             EXEDF   LOWCR   MCROOT
MCROV              EDCKP   EXEDF   GTMNM   LKLST   MCRDIS
PR1OV              ABOOV   EXEDF   FIXOV   GNBLK   GTMNM   GTTSK
                   LKLST   LN1OV   PR1OV
```

9.5 SYS GLOBAL CROSS-REFERENCEs

This cross-reference is for a mapped system.

The cross-reference contains an alphabetic listing of each global symbol along with its value and the name of each referencing module. When a symbol is defined in several segments within an overlay structure, TKB prints the last defined value in the listing. Similarly, in a real TKB cross-reference listing, TKB would print the module name more than once for each symbol if the module is loaded in several segments within the structure.

The Task Builder creates an SYS.CRF cross-reference file when /CR is specified in the Task Builder command file used to build SYS. One of the input files to the Task Builder when building SYS is the Executive symbol table file, RSX11M.STB. RSX11M.STB is needed because SYS references some Executive symbols. All the symbols from RSX11M.STB are put in the SYS.CRF symbol table file even though they are not referenced by SYS. Therefore, some symbols appearing here in the SYS cross-reference are defined in the Executive but not used by SYS. These symbols are shown defined in the Executive LOWCR or EXEDF modules.

The value contains the suffix -R if the symbol is relocatable.

Prefix symbols accompanying each module name define the type of reference as follows:

Prefix Symbol	Reference Type
blank	Module contains a reference that is resolved in the same segment or in a segment toward the root.
^	Module contains a reference that is resolved directly in a segment away from the root or in a co-tree.
@	Module contains a reference that is resolved through an autoload vector.
#	Module contains a reference that is resolved This module defines the symbol.
*	Module contains an autoloadable definition. This module defines the symbol.

```
Symbol  Value     Modules That Reference Symbol

DV.F11  040000    # EXEDF    SDSOV     SPROV
DV.ISP  002000    # EXEDF    REDOV
DV.MNT  100000      ALLOV    DEAOV     DEVOV    # EXEDF     REDOV
                    SDSOV    SPROV
DV.OSP  004000    # EXEDF    REDOV
DV.PSE  010000      ALLOV    ASNOV     DEAOV      DEVOV   # EXEDF
                  # LOWCR    OPEOV     REDOV      SDSOV     SPROV
DV.SQD  000040      DEVOV  # EXEDF
DV.TTY  000004      ALLOV    ASNOV     ATLOV      DEVOV   # EXEDF
                  # LOWCR    SDSOV     SPROV
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

DV.UMD    000200    # EXEDF     LOWCR
D$$YNM    000000    # LOWCR
D.DSP     000012      DEVOV   # EXEDF   # LOWCR
D.LNK     000000      DEAOV     DEVOV   # EXEDF     SDSOV     $FDUCB
D.MSK     000014    # EXEDF   # LOWCR
D.NAM     000004      ASNOV     DEAOV     DEVOV   # EXEDF     FMTDV
                      GTTSK   # LOWCR     OPEOV     RAPOV     REDOV
                      SPROV     $FDUCB
D.PCB     000034      DEVOV   # EXEDF   # LOWCR     OPEOV
D.RS00    000000    # LOWCR
D.RS1     177777    # LOWCR
D.RS10    177766    # LOWCR
D.RS16    177760    # LOWCR
D.RS17    177757    # LOWCR
D.RS19    177755    # LOWCR
D.RS2     177776    # LOWCR
D.RS22    000002    # LOWCR
D.RS5     177773    # LOWCR
D.RS6     177772    # LOWCR
D.RS7     177771    # LOWCR
D.RS8     177770    # LOWCR
D.RS80    177660    # LOWCR
D.RS81    177657    # LOWCR
D.RS84    177654    # LOWCR
D.RS85    177653    # LOWCR
D.RS86    177652    # LOWCR
D.RS87    177651    # LOWCR
D.RS90    177646    # LOWCR
D.RS92    177644    # LOWCR
D.RS93    177643    # LOWCR
D.RS94    177642    # LOWCR
D.RS95    177641    # LOWCR
D.RS96    177640    # LOWCR
D.RS97    177637    # LOWCR
D.RS98    177636    # LOWCR
D.RS99    177635    # LOWCR
D.UCB     000002      DEAOV     DEVOV   # EXEDF     FMTDV     GTMNM
                    # LOWCR     OPEOV     SDSOV     $FDUCB
D.UCBL    000010      DEAOV     DEVOV   # EXEDF     FMTDV     GTMNM
                    # LOWCR     SDSOV     $FDUCB
D.UNIT    000006      DEAOV     DEVOV   # EXEDF     FMTDV     GTMNM
                    # LOWCR     $FDUCB
D.VCAN    000002    # EXEDF   # LOWCR
D.VINI    000000    # EXEDF   # LOWCR
D.VOUT    000004    # EXEDF   # LOWCR
D.VPWF    000006    # EXEDF   # LOWCR
EC.DTO    000140    # EXEDF   # LOWCR
EC.DVC    000001    # EXEDF   # LOWCR
EC.NSI    000141    # EXEDF   # LOWCR
E.LGTH    000056    # EXEDF   # LOWCR
E.OPC     000022    # EXEDF   # LOWCR
E.RTRY    000016    # EXEDF   # LOWCR
E.SIZE    000000    # EXEDF   # LOWCR
FE.DRV    000010      DEVOV   # EXEDF
FE.EXP    000200    # EXEDF     SDSOV     SPROV
FE.EXT    000001    # EXEDF     PAROV     SDSOV     SETOV     TASOV
FE.EXV    000004    # EXEDF     SPROV
FE.MUP    000002      ALLOV     DEAOV     DEVOV   # EXEDF     RPSOV
                      RUNOV     SPROV
FE.NLG    100000    # EXEDF     SPROV
FE.PKT    000100    # EXEDF     SDSOV     SPROV
FE.PLA    000020    # EXEDF     SETOV     SPROV
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

H.CSP     000000      ATLOV   # EXEDF
H.GARD    000072      ATLOV   # EXEDF
H.WND     000044    # EXEDF     SYSROT
IE.ABO    177761    # LOWCR
IE.ALN    177736    # LOWCR
IE.BAD    177777    # LOWCR
IE.BLK    177754    # LOWCR
IE.BYT    177755    # LOWCR
IE.DAA    177770    # LOWCR
IE.DNA    177771    # LOWCR
IE.DNR    177775    # LOWCR
IE.IFC    177776    # LOWCR
IE.LCK    177745    # LOWCR
IE.NLN    177733    # LOWCR
IE.NOD    177751    # LOWCR
IE.OFL    177677    # LOWCR
IE.OVR    177756    # LOWCR
IE.PRI    177760    # LOWCR
IE.SDP    177635      ALTOV
IE.SPC    177772    # LOWCR
IE.ULK    177653    # LOWCR
IE.UPN    177777      ASNOV
IO.ATT    001400      ASNOV     ATLOV     CLQOV     DEVOV   # LOWCR
                      OPEOV     PAROV     TASOV
IO.CLN    003400    # LOWCR
IO.DET    002000      ASNOV     ATLOV     CLQOV     DEVOV   # LOWCR
                      OPEOV     PAROV     TASOV
IO.KIL    000012      ERROV
IO.LOV    001010    # LOWCR
IO.RLB    001000    # LOWCR     RAPOV     TASOV
IO.RVB    010400    # LOWCR     OPEOV
IO.ULK    005000    # LOWCR
IO.WAL    000410      SPROV
IO.WLB    000400    # LOWCR     OPEOV     REAOV
IO.WVB    011000      ASNOV     ATLOV     CLQOV     DEVOV     ERROV
                    # LOWCR     OPEOV     PAROV     SDSOV     SPROV
                      TASOV     TIMOV
IQ.UMD    000004    # LOWCR
IS.SUC    000001    # LOWCR
KISAR5    172352    # EXEDF   # LOWCR
KISAR6    172354    # EXEDF   # LOWCR
L.ASG     000010      ASNOV   # EXEDF   # LOWCR    $FDUCB
L.LGTH    000012      ASNOV   # EXEDF
L.LNK     000000      ASNOV   # EXEDF    $FDUCB
L.NAM     000002      ASNOV   # EXEDF   # LOWCR    $FDUCB
L.TYPE    000005      ASNOV   # EXEDF   # LOWCR    $FDUCB
L.UCB     000006      ASNOV   # EXEDF   # LOWCR    $FDUCB
L.UNIT    000004      ASNOV   # EXEDF   # LOWCR    $FDUCB
M$$MGE    000000    # LOWCR
PADVBF    125730-R    DEVOV   # PAROV
PR7       000340    # EXEDF     SETOV
PS        177776    # EXEDF     SETOV
P.ATT     000036    # LOWCR
P.BLKS    000016    # EXEDF   # LOWCR
P.BUSY    000024    # EXEDF   # LOWCR
P.HDR     000032      ATLOV   # LOWCR     OPEOV
P.IOC     000003    # EXEDF   # LOWCR
P.LGTH    000042    # LOWCR     SETOV
P.LNK     000000    # EXEDF   # LOWCR
P.MAIN    000012      ALTOV   # EXEDF   # LOWCR
P.NAM     000004    # EXEDF   # LOWCR
P.OWN     000026    # EXEDF   # LOWCR
```

```
                           CROSS-REFERENCES

Symbol  Value       Modules That Reference Symbol

P.PRI     000002    # EXEDF   # LOWCR
P.PRO     000034    # LOWCR
P.REL     000014    # EXEDF   # LOWCR
P.SIZE    000016    # EXEDF   # LOWCR
P.STAT    000030    # EXEDF   # LOWCR
P.SUB     000010    # EXEDF   # LOWCR
P.SWSZ    000022    # EXEDF   # LOWCR
P.TCB     000026    # EXEDF   # LOWCR
P.WAIT    000020      ALTOV   # EXEDF   # LOWCR
SP.EIP    000001    # EXEDF   # LOWCR
SP.ENB    000002    # EXEDF   # LOWCR
S.BMSK    177776    # EXEDF   # LOWCR
S.BMSV    177774    # EXEDF   # LOWCR
S.CCB     000030    # LOWCR
S.CON     000010    # EXEDF   # LOWCR
S.CSR     000012    # EXEDF   # LOWCR     SETOV
S.CTM     000006    # EXEDF   # LOWCR
S.DZCK    000030    # LOWCR
S.FRK     000016    # EXEDF   # LOWCR
S.ITM     000007    # EXEDF   # LOWCR
S.LHD     000000    # EXEDF   # LOWCR
S.MPR     000030    # LOWCR
S.PKT     000014    # EXEDF   # LOWCR
S.PRI     000004    # EXEDF   # LOWCR
S.RCNT    177772    # EXEDF   # LOWCR
S.ROFF    177773    # EXEDF   # LOWCR
S.STS     000011    # EXEDF   # LOWCR
S.VCT     000005    # EXEDF   # LOWCR
T.EXT     000000    # LOWCR
T.LGTH    000070    # LOWCR
T.NAM     000006      CLQOV   # EXEDF     PAROV
T.PCB     000046    # EXEDF     LKLST     SETOV
T.ST2     000034    # EXEDF     LKLST
T.ST3     000036    # EXEDF     RPSOV     SYSOV
T.TCBL    000030    # EXEDF     SETOV
T.UCB     000026    # EXEDF     GTMNM     GTTSK     SPROV     SYSOV
                      $FDUCB
T2.CKD    010000    # EXEDF     LKLST
T3.MCR    004000    # EXEDF     RPSOV     SYSOV
UC.ATT    000010    # EXEDF   # LOWCR
UC.KIL    000004    # EXEDF   # LOWCR
UC.LGH    000003    # EXEDF   # LOWCR
UC.NPR    000100    # EXEDF   # LOWCR
UC.PWF    000020    # EXEDF   # LOWCR
UC.QUE    000040    # EXEDF   # LOWCR
UISAR0    177640    # EXEDF   # LOWCR
UISDR0    177600    # EXEDF   # LOWCR
US.BSY    000200    # EXEDF   # LOWCR
US.FOR    000040    # EXEDF   # LOWCR
US.MDM    000020      DEVOV   # EXEDF   # LOWCR
US.MNT    000100      ALLOV     DEAOV     DEVOV   # EXEDF   # LOWCR
                      REDOV
US.OFL    000001      DEVOV   # EXEDF   # LOWCR     SETOV
US.PUB    000004      ALLOV     DEAOV     DEVOV   # EXEDF   # LOWCR
                      SDSOV     SPROV
US.RED    000002    # EXEDF     REDOV
US.UMD    000010    # EXEDF   # LOWCR
US.WCK    000010    # EXEDF     SDSOV     SPROV
U.ACP     000032    # EXEDF   # LOWCR
U.ATT     000022      ALLOV     ASNOV     ATLOV     CLQOV     DEVOV
                    # EXEDF   # LOWCR     REDOV
U.BUF     000024    # EXEDF   # LOWCR
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

U.CNT     000030    # EXEDF   # LOWCR
U.CTL     000004    # EXEDF   # LOWCR
U.CW1     000010      ALLOV     ASNOV     ATLOV     DEAOV     DEVOV
                    # EXEDF   # LOWCR     OPEOV     REDOV     SDSOV
                      SPROV
U.CW2     000012      ALLOV     ASNOV     ATLOV     DEAOV     DEVOV
                    # EXEDF   # LOWCR     RUNOV     SDSOV     SETOV
                      SPROV     SYSOV
U.CW3     000014    # EXEDF   # LOWCR     SDSOV     SETOV
U.CW4     000016    # EXEDF     SDSOV     SPROV
U.DCB     000000    # EXEDF     FMTDV     GTMNM     GTTSK     REDOV
                      SDSOV     $FDUCB
U.LUIC    177774      DEVOV   # EXEDF     RUNOV     SPROV
U.OWN     177776      ALLOV     DEAOV     DEVOV   # EXEDF   # LOWCR
                      SDSOV     SPROV
U.RED     000002      ASNOV     DEVOV   # EXEDF     GTMNM   # LOWCR
                      REDOV     SPROV
U.SCB     000020    # EXEDF   # LOWCR     SETOV
U.STS     000005      ALLOV     DEAOV     DEVOV   # EXEDF   # LOWCR
                      REDOV     SDSOV     SPROV
U.ST2     000007      ALLOV     DEAOV     DEVOV   # EXEDF   # LOWCR
                      REDOV     SDSOV     SETOV     SPROV
U.UIC     000052    # EXEDF     SDSOV     SPROV
U.UNIT    000006    # EXEDF   # LOWCR     SETOV
U.VCB     000034      DEVOV   # EXEDF   # LOWCR
U2.CRT    002000    # EXEDF     SDSOV     SPROV
U2.DH1    100000    # EXEDF     SPROV
U2.DJ1    040000    # EXEDF     SPROV
U2.DZ1    000100    # EXEDF     SDSOV     SETOV     SPROV
U2.ESC    001000    # EXEDF     SDSOV     SPROV
U2.HLD    000040    # EXEDF     SDSOV     SPROV
U2.LOG    000400      ALLOV     DEVOV   # EXEDF
U2.LWC    000001    # EXEDF     SDSOV     SPROV
U2.L3S    000004    # EXEDF     SDSOV     SPROV
U2.L8S    010000    # EXEDF     SDSOV     SPROV
U2.PRV    000010      ASNOV     ATLOV     DEAOV   # EXEDF    RUNOV
                      SDSOV     SPROV     SYSOV
U2.RMT    020000    # EXEDF     SDSOV     SPROV
U2.SLV    000200    # EXEDF     SDSOV     SPROV
U2.VT5    000002    # EXEDF     SDSOV     SPROV
V$$CTR    000410    # LOWCR
W.BLVR    000002    # EXEDF     SYSROT
X.AST     000032    # LOWCR
X.DSI     000024    # LOWCR
X.FORK    000012    # LOWCR
X.ISR     000010    # LOWCR
X.JSR     000002    # LOWCR
X.LEN     000050    # LOWCR
X.LNK     000000    # LOWCR
X.PSW     000006    # LOWCR
X.REL     000022    # LOWCR
X.TCB     000026    # LOWCR
X.VEC     000044    # LOWCR
X.VPC     000046    # LOWCR
$ABCTK    014460    # LOWCR
$ABTIM    005414      CLQOV   # LOWCR
$ABTSK    014464    # LOWCR
$ACHCK    007242    # LOWCR
$ACHKB    007250    # LOWCR
$ACHKP    007206    # LOWCR
$ACHKW    007232    # LOWCR
$ACTEP    122566-R  # ATLOV
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

$ACTHD    005634    # LOWCR
$ACTRM    015172    # LOWCR
$ACTTK    014652    # LOWCR
$ALCLK    006636    # LOWCR
$ALEB1    032634    # LOWCR
$ALEMB    032620    # LOWCR
$ALLEP    123350-R  # ALLOV     REAOV
$ALOCB    006524      ASNOV   # LOWCR     RPSOV     SETOV
$ALOC1    006566    # LOWCR
$ALPKT    006652    # LOWCR
$ALTEP    122170-R  # ALTOV
$ASNEP    122636-R  # ASNOV     RAPOV
$ATLEP    122556-R  # ATLOV
$BILDS    014540    # LOWCR
$BLKCK    010174    # LOWCR
$BLKC1    010204    # LOWCR
$BLXIO    006212    # LOWCR
$BMSET    032730    # LOWCR
$BRKEP    124642-R  # BRKOV     OPEOV
$BTMSK    005640    # LOWCR
$CAT5     127134      ALTOV     GTTSK     OPEOV     RAPOV     RPSOV
                      SPROV     SYSOV
$CBDMG    124206      ATLOV     CLQOV     SDSOV     TASOV
$CBOMG    125304      ASNOV     ATLOV     FMTDV     GTMNM     SDSOV
                      TASOV
$CDTB     127310      GETNUM    RPSOV     TIMOV
$CEFI     007362    # LOWCR
$CEFN     007356    # LOWCR
$CFLPT    005522    # LOWCR
$CHKPT    016226    # LOWCR
$CKACC    033726    # LOWCR
$CKCNT    005604    # LOWCR
$CKCSR    005606    # LOWCR
$CKINT    017726    # LOWCR
$CKLDC    005610    # LOWCR
$CLINS    014162    # LOWCR     RUNOV
$CLKHD    005556      CLQOV   # LOWCR
$CLRMV    014264    # LOWCR
$COMEF    005570    # LOWCR
$COPT     005560    # LOWCR
$COTB     127316      ASNOV     GETNUM    OPEOV     $FDUCB
$CRASH    001470    # LOWCR
$CRATT    034024    # LOWCR
$CRAVL    005532    # LOWCR     SDSOV
$CRPAS    012470    # LOWCR
$CRSBF    000730    # LOWCR
$CRSBN    001462    # LOWCR
$CRSCS    001466    # LOWCR
$CRSHT    001752    # LOWCR
$CRSST    001460    # LOWCR
$CRSUN    001756    # LOWCR
$CVRTM    007070    # LOWCR
$C5TA     005772      ATLOV     CLQOV     ERROV   # LOWCR     PAROV
                      SDSOV     TASOV
$DASTT    015030    # LOWCR
$DB0      043366    # LOWCR
$DDIV     124454      CLQOV
$DEACB    006672      ASNOV   # LOWCR     RPSOV     RUNOV     SETOV
                      SPROV
$DEAC1    006732    # LOWCR
$DEAEP    123624-R  # DEAOV     REAOV
$DECLK    006644    # LOWCR
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

$DEPKT    006666     # LOWCR
$DETRG    031502     # LOWCR
$DEVEP    122302-R   # DEVOV
$DEVHD    005462       DEAOV      DEVOV    # LOWCR      OPEOV      SDSOV
                       $FDUCB
$DEVTB    043122     # LOWCR
$DIRSV    002264     # LOWCR
$DIRXT    002514     # LOWCR
$DIV      012564       FMTDV      GTMNM    # LOWCR      TIMOV
$DK0      043630     # LOWCR
$DPLM1    021662     # LOWCR
$DPLM2    021666     # LOWCR
$DQLM1    026526     # LOWCR
$DQLM2    026536     # LOWCR
$DRABO    022140     # LOWCR
$DRASG    022164     # LOWCR
$DRATP    032020     # LOWCR
$DRATR    031166     # LOWCR
$DRATX    022410     # LOWCR
$DRCEF    032340     # LOWCR
$DRCMT    022556     # LOWCR
$DRCRR    030600     # LOWCR
$DRCRW    024064     # LOWCR
$DRCSR    022562     # LOWCR
$DRDAR    022576     # LOWCR
$DRDCP    022632       EDCKP    # LOWCR
$DRDSE    032350       LKLST    # LOWCR
$DRDTR    031350     # LOWCR
$DREAR    022612     # LOWCR
$DRECP    022656       EDCKP    # LOWCR
$DREIF    004220     # LOWCR
$DRELW    024312     # LOWCR
$DREXP    022702     # LOWCR
$DREXT    004226     # LOWCR
$DRFEX    026100     # LOWCR
$DRGCL    023372     # LOWCR
$DRGLI    023500     # LOWCR
$DRGMX    025504     # LOWCR
$DRGPP    023602     # LOWCR
$DRGSS    023704     # LOWCR
$DRGTK    023714     # LOWCR
$DRGTP    024030     # LOWCR
$DRLM1    021566     # LOWCR
$DRLM2    021606     # LOWCR
$DRMAP    024346     # LOWCR
$DRMKT    025730     # LOWCR
$DRPUT    026122     # LOWCR
$DRQIO    026410     # LOWCR
$DRQRQ    027516     # LOWCR
$DRRAF    032364     # LOWCR
$DRRCV    026162     # LOWCR
$DRREC    030306     # LOWCR
$DRREQ    031714     # LOWCR
$DRRES    031746     # LOWCR
$DRRRA    026142     # LOWCR
$DRRRF    025232     # LOWCR
$DRRUN    025762     # LOWCR
$DRSDV    032532     # LOWCR
$DRSEF    032420     # LOWCR
$DRSND    030440     # LOWCR
$DRSPN    032006     # LOWCR
$DRSRF    024716     # LOWCR
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

$DRSTV    032540    # LOWCR
$DRUNM    024672    # LOWCR
$DRWFL    032450    # LOWCR
$DRWFS    032512    # LOWCR
$DRWSE    032434    # LOWCR
$DSW      000046      ALTOV     ASNOV
$DS0      044064    # LOWCR
$DTOER    032754    # LOWCR
$DT0      044276    # LOWCR
$DVCER    033016    # LOWCR
$DVERR    033016    # LOWCR
$DVMSG    007446    # LOWCR
$DX0      044472    # LOWCR
$DYPMN    005470    # LOWCR     TIMOV
$EMSST    017156    # LOWCR
$EMTRP    021512    # LOWCR
$ERREP    122160-R  # ERROV
$ERRHD    005700    # LOWCR
$ERRLM    005704    # LOWCR
$ERRLN    000404    # ERRMSG    ERROV
$ERRPT    005514    # LOWCR
$ERRSQ    005706    # LOWCR
$ERRSV    005710    # LOWCR
$ERRSZ    005712    # LOWCR
$ERRTB    124364-R  # ERRMSG    ERROV
$EXRQF    016434    # LOWCR     RPSOV
$EXRQN    016452    # LOWCR
$EXRQP    016426    # LOWCR
$EXSIZ    005520    # LOWCR     SDSOV     SETOV     SPROV
$FDLGG    126422-R  # $FDUCB
$FDLOG    126430-R    ALLOV     ASNOV     DEAOV     RAPOV     SPROV
                    # $FDUCB
$FDUCB    126436-R    ASNOV     ATLOV     REDOV   # $FDUCB
$FINBF    003776    # LOWCR
$FLTRP    017212    # LOWCR
$FMASK    005552      ALLOV     DEAOV     DEVOV   # LOWCR     OPEOV
                      PAROV     RPSOV     RUNOV     SDSOV     SETOV
                      SPROV     TASOV
$FMTDV    125176-R    ASNOV     ATLOV     DEVOV   # FMTDV     SDSOV
                      TASOV
$FNDSP    015750    # LOWCR
$FORK     002320    # LOWCR
$FORK0    002342    # LOWCR
$FORK1    002340    # LOWCR
$FPINT    017226    # LOWCR
$FRKHD    005546    # LOWCR
$GNBLK    126750-R    ALLOV     ASNOV     ATLOV     DEAOV     DEVOV
                    # GNBLK
$GTBYT    006102    # LOWCR
$GTMNM    126034-R  # GTMNM     GTTSK
$GTNUM    126716-R    ALTOV   # GETNUM    RAPOV     RPSOV     SPROV
$GTPKT    007510    # LOWCR
$GTTSK    125460-R    ALTOV   # GTTSK     RAPOV
$GTWRD    006160    # LOWCR
$HEADR    005564      GTTSK     LKLST   # LOWCR     OPEOV     RAPOV
                      RPSOV     RUNOV     SETOV     SYSROT    TASOV
                      $FDUCB
$ICHKP    016160    # LOWCR
$ILINS    017304    # LOWCR
$INITL    052414    # LOWCR
$INTCT    005542    # LOWCR
$INTSE    002376    # LOWCR
```

CROSS-REFERENCES

```
Symbol    Value        Modules That Reference Symbol

$INTSV    002434     # LOWCR
$INTXT    002374     # LOWCR
$INTX1    002456     # LOWCR
$IOABM    005714     # LOWCR
$IOALT    010272     # LOWCR
$IODON    010274     # LOWCR
$IOFIN    010414     # LOWCR
$IOKIL    010666     # LOWCR
$IOTRP    017316     # LOWCR
$LCKPR    011552     # LOWCR
$LDPWF    014124     # LOWCR
$LDRPT    005464     # LOWCR
$LOAD     121004       SYSROT
$LOADR    041214     # LOWCR
$LOADT    016422     # LOWCR
$LOCKL    124564-R     ASNOV       ATLOV       DEVOV     # LKLST       PAROV
                       REDOV       SETOV       TASOV
$LOGHD    005624       ASNOV     # LOWCR       $FDUCB
$LPO      044614     # LOWCR
$LSTLK    005630       LKLST     # LOWCR
$MAPTK    017036     # LOWCR
$MBUF     120362-R     ASNOV       ATLOV       CLQOV       DEVOV       ERROV
                       OPEOV       PAROV       RPSOV       SDSOV       SPROV
                       SYSOV     # SYSROT      TASOV       TIMOV
$MCKD     122512-R   # EDCKP       ERROV
$MCKE     122524-R   # EDCKP       ERROV
$MCMD     120454-R     ERROV       RAPOV       RPSOV       SPROV       SYSOV
                     # SYSROT
$MCOV     120464-R     SPROV       SYSOV     # SYSROT
$MCR      120634-R     ALLOV       ALTOV       ASNOV       ATLOV       BRKOV
                       CLQOV       DEAOV       DEVOV       ERROV       OPEOV
                       PAROV       REAOV       REDOV       RPSOV       RUNOV
                       SDSOV       SETOV       SPROV     # SYSROT      TASOV
                       TIMOV
$MCRCB    005626     # LOWCR
$MCRPT    005466     # LOWCR       RPSOV
$MDIS     120500-R   # SYSROT
$MDPB     120366-R     ASNOV       ATLOV       DEVOV       ERROV       OPEOV
                       PAROV       RAPOV       REAOV       SDSOV       SPROV
                     # SYSROT      TASOV       TIMOV
$MERLD    120720-R     ALLOV       ALTOV       ASNOV       ATLOV       CLQOV
                       DEAOV       DEVOV       OPEOV       PAROV       RAPOV
                       REAOV       REDOV       RPSOV       RUNOV       SDSOV
                       SETOV       SPROV       SYSOV     # SYSROT      TASOV
                       TIMOV
$MERR     120470-R   # SYSROT
$MERRN    120474-R     ALLOV       ALTOV       ASNOV       ATLOV       CLQOV
                       DEAOV       DEVOV       ERROV       OPEOV       PAROV
                       RAPOV       REAOV       REDOV       RPSOV       RUNOV
                       SDSOV       SETOV       SPROV       SYSOV     # SYSROT
                       TASOV       TIMOV
$MLDOV    120724-R     RAPOV       RPSOV       SPROV       SYSOV     # SYSROT
$MLIMI    120632-R     ATLOV       DEVOV       PAROV     # SYSROT      TASOV
$MLINE    120504-R     SYSOV     # SYSROT
$MMO      045014     # LOWCR
$MOVRB    120416-R   # SYSROT
$MPARS    120424-R     ALLOV       ALTOV       ASNOV       ATLOV       DEAOV
                       DEVOV       ERROV       OPEOV       PAROV       RAPOV
                       REAOV       REDOV       RPSOV       RUNOV       SDSOV
                       SETOV       SPROV     # SYSROT      TASOV       TIMOV
$MPCSR    012734     # LOWCR
$MPLND    011126     # LOWCR
```

```
Symbol   Value       Modules That Reference Symbol

$MPLNE    011106     # LOWCR
$MPLUN    011070     # LOWCR
$MPPHY    011170     # LOWCR
$MPPKT    011240     # LOWCR
$MPRSR    120460-R     SYSOV   # SYSROT
$MPVBN    011372     # LOWCR
$MROOT    120706-R   # SYSROT
$MTERM    120476-R     RPSOV     RUNOV     SYSOV   # SYSROT
$MUCB     120364-R     ALLOV     ASNOV     ATLOV     CLQOV     DEAOV
                       DEVOV     PAROV     RPSOV     RUNOV     SDSOV
                       SPROV     SYSOV   # SYSROT    TIMOV
$MUL      012534     # LOWCR     RUNOV
$MXEXT    005770     # LOWCR     SDSOV     SPROV
$NLO      051622     # LOWCR
$NNBLK    126752-R   # GNBLK     SPROV     TIMOV
$NONSI    002556     # LOWCR
$NS0      033300     # LOWCR
$NS1      033306     # LOWCR
$NS2      033314     # LOWCR
$NS3      033322     # LOWCR
$NS4      033330     # LOWCR
$NS5      033336     # LOWCR
$NS6      033344     # LOWCR
$NS7      033352     # LOWCR
$NULL     043100     # LOWCR
$NXTSK    015322       ALTOV   # LOWCR
$OPEEP    122226-R   # OPEOV
$PANIC    001470     # LOWCR
$PAREP    123654-R     DEVOV   # PAROV
$PARHD    005422     # LOWCR     OPEOV     PAROV     SDSOV     SETOV
                       SPROV
$PARPT    005554     # LOWCR
$PARTB    012670     # LOWCR
$PCBS     052164     # LOWCR
$PKAVL    005764     # LOWCR
$PKMAX    005767     # LOWCR     SDSOV     SPROV
$PKNUM    005766     # LOWCR     SDSOV
$POOL     052310     # LOWCR
$POWER    013750     # LOWCR
$PP0      045144     # LOWCR
$PR0      045266     # LOWCR
$PTBYT    006132     # LOWCR
$PTWRD    006160     # LOWCR
$PWRFL    005416     # LOWCR
$QASTT    015062     # LOWCR
$QEMB     033516     # LOWCR
$QINSF    014332     # LOWCR
$QINSP    014340       ALTOV   # LOWCR
$QMCRL    014376     # LOWCR
$QRMVF    014406     # LOWCR
$QRMVT    014420       ALTOV   # LOWCR
$RAPEP    122144-R   # RAPOV
$REAEP    123142-R   # REAOV
$REDEP    123000-R     ALTOV   # REDOV
$RELOC    012222     # LOWCR
$RELOM    012272     # LOWCR
$RELOP    012410     # LOWCR
$RLMCB    023434     # LOWCR
$RLPAR    015246     # LOWCR
$RLPR1    015310     # LOWCR
$RPSEP    122206-R   # RPSOV
$RQSCH    005452     # LOWCR
```

```
Symbol  Value       Modules That Reference Symbol

$RUNEP    122144-R  # RUNOV
$SAVNR    004144      CBTO      CBTO    # LOWCR
$SCDVT    012306    # LOWCR
$SCDV1    012312    # LOWCR
$SDSEP    122676-R  # SDSOV
$SETCR    014704    # LOWCR
$SETEP    122164-R  # SETOV
$SETF     014762    # LOWCR
$SETM     014766    # LOWCR
$SETRQ    014734    # LOWCR
$SETRT    014732    # LOWCR
$SGFLT    017336    # LOWCR
$SHFPT    005516    # LOWCR
$SIGFL    005420      LKLST   # LOWCR
$SOVEP    122134-R  # SYSOV   ^ SYSROT
$SPREP    122700-R  # SPROV
$SRATT    034124    # LOWCR
$SRNAM    033644    # LOWCR
$SRSTD    015132      GTTSK   # LOWCR     OPEOV     RPSOV     RUNOV
$SRWND    034162    # LOWCR
$STACK    000642    # LOWCR
$STD      052220    # LOWCR
$STKDP    005454    # LOWCR     RUNOV
$STPCT    015222    # LOWCR
$STPTK    015226    # LOWCR
$SWSTK    004164    # LOWCR
$SYBEG    053424    # LOWCR
$SYSID    005574    # LOWCR
$SYSIZ    005716    # LOWCR     OPEOV     SETOV     SPROV
$SYTOP    063424    # LOWCR
$SYUIC    005612    # LOWCR     SDSOV     SPROV
$TALEP    122576-R  # ATLOV
$TASEP    122172-R  # TASOV
$TIMEP    122260-R  # TIMOV
$TKNPT    005600    # LOWCR
$TKPS     005744      CLQOV   # LOWCR     RUNOV     TIMOV
$TKTCB    005446      EDCKP     GTMNM     GTTSK     LKLST   # LOWCR
                      RPSOV     SPROV     SYSOV     $FDUCB
$TKWSE    032430      LKLST   # LOWCR
$TRACE    017376      BRKOV   # LOWCR
$TRP04    017410    # LOWCR
$TRTRP    021446    # LOWCR
$TSKHD    005512      ALTOV   # LOWCR     SETOV     TASOV
$TSKRP    016504    # LOWCR     RUNOV
$TSKRQ    016502    # LOWCR
$TSKRT    016476    # LOWCR
$TSTCP    016052    # LOWCR
$TTNS     005762      CLQOV   # LOWCR     RUNOV     TIMOV
$TT0      045534    # LOWCR
$TT1      045564    # LOWCR
$TT10     047672    # LOWCR
$TT11     047722    # LOWCR
$TT12     047752    # LOWCR
$TT13     050002    # LOWCR
$TT14     050032    # LOWCR
$TT15     050062    # LOWCR
$TT16     050112    # LOWCR
$TT17     050142    # LOWCR
$TT2      047452    # LOWCR
$TT20     050172    # LOWCR
$TT21     050222    # LOWCR
$TT22     051210    # LOWCR
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

$TT23     051260    # LOWCR
$TT24     051310    # LOWCR
$TT25     051340    # LOWCR
$TT26     051370    # LOWCR
$TT27     051420    # LOWCR
$TT3      047502    # LOWCR
$TT30     051450    # LOWCR
$TT31     051500    # LOWCR
$TT4      047532    # LOWCR
$TT5      047562    # LOWCR
$TT6      047612    # LOWCR
$TT7      047642    # LOWCR
$UISET    016776    # LOWCR
$UNLKL    124654-R    ALTOV     ASNOV     ATLOV     DEVOV   # LKLST
                      PAROV     REDOV     SETOV     SYSOV     TASOV
$UNMAP    034222    # LOWCR
$USRTB    000000    # LOWCR
$XDT      035002    # LOWCR
.CBTO     124752-R  # CBTO      DEVOV     OPEOV     PAROV
.CL0      051756    # LOWCR
.COT2B    125056-R  # COT2B     OPEOV
.CO0      051732    # LOWCR
.C2BTO    124732-R  # CBTO      OPEOV     PAROV
.C22TO    124666-R  # CBTO      OPEOV     PAROV
.DB0      043162    # LOWCR
.DB1      043222    # LOWCR
.DB2      043262    # LOWCR
.DB3      043322    # LOWCR
.DK0      043464    # LOWCR
.DK1      043524    # LOWCR
.DK2      043564    # LOWCR
.DSW      000044    # LOWCR
.DS0      043720    # LOWCR
.DS1      043760    # LOWCR
.DS2      044020    # LOWCR
.DT0      044162    # LOWCR
.DT1      044226    # LOWCR
.DX0      044366    # LOWCR
.DX1      044426    # LOWCR
.KEYWD    127020-R  # KEYWD     OPEOV     RPSOV     SPROV
.LB0      052002    # LOWCR
.LDR      052220    # LOWCR
.LDRHD    052040    # LOWCR
.LP0      044562    # LOWCR
.MM0      044704    # LOWCR
.MM1      044746    # LOWCR
.NL0      051570    # LOWCR
.PP0      045112    # LOWCR
.PR0      045234    # LOWCR
.SY0      052026    # LOWCR
.TI0      051706    # LOWCR
.TT0      045362    # LOWCR
.TT1      045452    # LOWCR
.TT10     046400    # LOWCR
.TT11     046470    # LOWCR
.TT12     046560    # LOWCR
.TT13     046650    # LOWCR
.TT14     046740    # LOWCR
.TT15     047030    # LOWCR
.TT16     047120    # LOWCR
.TT17     047210    # LOWCR
.TT2      045660    # LOWCR
```

```
                               CROSS-REFERENCES

Symbol  Value       Modules That Reference Symbol

.TT20     047300    # LOWCR
.TT21     047370    # LOWCR
.TT22     050316    # LOWCR
.TT23     050406    # LOWCR
.TT24     050476    # LOWCR
.TT25     050566    # LOWCR
.TT26     050656    # LOWCR
.TT27     050746    # LOWCR
.TT3      045750    # LOWCR
.TT30     051036    # LOWCR
.TT31     051126    # LOWCR
.TT4      046040    # LOWCR
.TT5      046130    # LOWCR
.TT6      046220    # LOWCR
.TT7      046310    # LOWCR
```

9.6 SYS SEGMENT CROSS-REFERENCES

The SYS segment cross-reference lists the name of each overlay and the modules that compose it. The cross-reference follows:

```
Segment
Name            Resident Modules

ALTOV             ALTOV   EXEDF   GETNUM  GTMNM   GTTSK   LKLST
                  REDOV   $FDUCB
ATLOV             ATLOV   EXEDF   FMTDV   GNBLK   GTMNM   GTTSK
                  LKLST   $FDUCB
CLQOV             CLQOV   EXEDF
DEVOV             CBTO    DEVOV   EXEDF   FMTDV   GNBLK   LKLST
                  PAROV
ERROV             EDCKP   ERRMSG  ERROV
OPEOV             BRKOV   CBTO    COT2B   KEYWD   OPEOV
RAPOV             ASNOV   EXEDF   FMTDV   GETNUM  GNBLK   GTMNM
                  GTTSK   LKLST   RAPOV   $FDUCB
REAOV             ALLOV   DEAOV   EXEDF   GNBLK   REAOV   $FDUCB
RPSOV             EXEDF   GETNUM  KEYWD   RPSOV
RUNOV             EXEDF   RUNOV
SDSOV             EXEDF   FMTDV   SDSOV
SETOV             EXEDF   LKLST   SETOV
SPROV             EXEDF   GETNUM  GNBLK   KEYWD   SPROV   $FDUCB
SYSOV             EXEDF   LKLST   SYSOV
SYSROT            EXEDF   LOWCR   SYSROT
TASOV             EXEDF   FMTDV   LKLST   TASOV
TIMOV             GNBLK   TIMOV
```

9.7 BIGFCP GLOBAL CROSS REFERENCES

The cross-reference contains an alphabetic listing of each global symbol along with its value and the name of each referencing module. When a symbol is defined in several segments within an overlay structure, TKB prints the last defined value in the listing. Similarly, in a real TKB cross-reference listing, TKB would print the module name more than once for each symbol if the module is loaded in several segments within the structure.

CROSS-REFERENCES

The value contains the suffix -R if the symbol is relocatable.

The Task Builder creates a BIGFCP.CRF cross-reference file when /CR is specified in the Task Builder command file used to build BIGFCP. One of the input files to the Task Builder when building BIGFCP is the Executive symbol table file, RSX11M.STB. RSX11M.STB is needed because BIGFCP references some Executive symbols. All the symbols from RSX11M.STB are put in the BIGFCP.CRF symbol table file even though they are not referenced by BIGFCP. Therefore, some symbols appearing here in the BIGFCP cross-reference are defined in the Executive but not used by BIGFCP. These symbols are shown defined in the Executive LOWCR or EXEDF modules.

Prefix symbols accompanying each module name define the type of reference as follows:

Prefix
Symbol   Reference Type

blank   Module contains a reference that is resolved in the same segment or in a segment toward the root.

^   Module contains a reference that is resolved directly in a segment away from the root or in a co-tree.

@   Module contains a reference that is resolved through an autoload vector.

Module contains a non-autoloadable definition. This module defines the symbol.

*   Module contains an autoloadable definition. This module defines the symbol.

```
Symbol  Value     Modules That Reference Symbol

AT.FCB   000100    # ATCTL     RATCM     WATCM
AT.HDR   000000    # ATCTL
AT.IDN   000001    # ATCTL
AT.MAP   000002    # ATCTL
AT.PRO   000040    # ATCTL     WATCM
AT.RO    000200    # ATCTL     WATCM
DV.PSE   010000    # EXEDF   # LOWCR
DV.TTY   000004    # EXEDF   # LOWCR
DV.UMD   000200    # EXEDF   # LOWCR
D$$YNM   000000    # LOWCR
D.DSP    000012    # EXEDF   # LOWCR
D.MSK    000014    # EXEDF   # LOWCR
D.NAM    000004    # EXEDF   # LOWCR
D.PCB    000034    # EXEDF   # LOWCR
D.RS00   000000    # LOWCR
D.RS1    177777    # LOWCR
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

D.RS10    177766    # LOWCR
D.RS16    177760    # LOWCR
D.RS17    177757    # LOWCR
D.RS19    177755    # LOWCR
D.RS2     177776    # LOWCR
D.RS22    000002    # LOWCR
D.RS5     177773    # LOWCR
D.RS6     177772    # LOWCR
D.RS7     177771    # LOWCR
D.RS8     177770    # LOWCR
D.RS80    177660    # LOWCR
D.RS81    177657    # LOWCR
D.RS84    177654    # LOWCR
D.RS85    177653    # LOWCR
D.RS86    177652    # LOWCR
D.RS87    177651    # LOWCR
D.RS90    177646    # LOWCR
D.RS92    177644    # LOWCR
D.RS93    177643    # LOWCR
D.RS94    177642    # LOWCR
D.RS95    177641    # LOWCR
D.RS96    177640    # LOWCR
D.RS97    177637    # LOWCR
D.RS98    177636    # LOWCR
D.RS99    177635    # LOWCR
D.UCB     000002    # EXEDF   # LOWCR
D.UCBL    000010    # EXEDF   # LOWCR
D.UNIT    000006    # EXEDF   # LOWCR
D.VCAN    000002    # EXEDF   # LOWCR
D.VINI    000000    # EXEDF   # LOWCR
D.VOUT    000004    # EXEDF   # LOWCR
D.VPWF    000006    # EXEDF   # LOWCR
EC.DTO    000140    # EXEDF   # LOWCR
EC.DVC    000001    # EXEDF   # LOWCR
EC.NSI    000141    # EXEDF   # LOWCR
E.BDHD    000000    # DISPAT    RDHDR
E.LGTH    000056    # EXEDF   # LOWCR
E.OPC     000022    # EXEDF   # LOWCR
E.RTRY    000016    # EXEDF   # LOWCR
E.SIZE    000000    # EXEDF   # LOWCR
FE.MUP    000002      CRFIL   # EXEDF     PROCK
F.EFBK    000010      DRACC     DREOF
F.FFBY    000014      DRACC     DREOF
F.HIBK    000004      DRACC     DREOF
F.RSIZ    000002      DRACC
F.RTYP    000000      DRACC
IE.ABO    177761    # LOWCR     RWVBL
IE.ALC    177654      SMALC
IE.ALN    177736    # LOWCR
IE.BAD    177777      ACCESS    CRFIL     DEACC     DISPAT    DLMRK
                      DRINI     ENTNM     EXCOM     GTFID     INWIN
                      LOCAT   # LOWCR     RWATT     RWVB
IE.BDR    177716      DRACC
IE.BHD    177700      NXHDR     RDHDR     SMDEL     SMSCN
IE.BLK    177754    # LOWCR
IE.BVR    177701      ENTNM
IE.BYT    177755    # LOWCR
IE.CKS    177742      RDHDR
IE.CLO    177732      ACCESS
IE.DAA    177770    # LOWCR
IE.DFU    177750      SMALC
IE.DNA    177771    # LOWCR
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

IE.DNR    177775    # LOWCR
IE.DUP    177707      ENTNM
IE.EOF    177766      RWVB      RWVBL     TRUNC
IE.HFU    177744      EXCMP
IE.IFC    177776    # LOWCR
IE.IFU    177747      CRFID
IE.LCK    177745      ACCESS  # LOWCR     TRUNC     WACCK
IE.NLN    177733      GTFID   # LOWCR
IE.NOD    177751      ALLOC   # LOWCR     RWVBL
IE.NSF    177746      DLMRK     FDRMV     PROCK     RDHDR
IE.OFL    177677    # LOWCR
IE.OVR    177756    # LOWCR
IE.PRI    177760    # LOWCR     PROCK     TRUNC     WRATT
IE.RER    177740      RW1LB
IE.SNC    177735      RDHDR
IE.SPC    177772    # LOWCR
IE.SQC    177734      NXHDR     RDHDR
IE.ULK    177653    # LOWCR
IE.UPN    177777      RWVBL     RW1LB
IE.WAC    177743      ACCESS
IE.WAT    177741      RATCM     WATCM
IE.WER    177737      CLNUP     RW1LB
IE.WLK    177764      CRFIL     PROCK
IO.ACR    006400      ACCESS
IO.ATT    001400    # LOWCR
IO.CLN    003400    # LOWCR
IO.CRE    012000      CLCOM
IO.DEL    012400      CLCRE     DLFIL
IO.DET    002000    # LOWCR
IO.EXT    011400      DREXT     IXEXT
IO.FNA    004400      DRACC
IO.LOV    001010    # LOWCR
IO.RLB    001000    # LOWCR     RWVBL     RW1LB
IO.RVB    010400    # LOWCR
IO.ULK    005000    # LOWCR
IO.WLB    000400    # LOWCR     RWVBL     RW1LB
IO.WVB    011000    # LOWCR     RWVBL
IQ.UMD    000004    # LOWCR
IS.SUC    000001    # LOWCR
KISAR5    172352    # EXEDF   # LOWCR
KISAR6    172354    # EXEDF   # LOWCR
L.ASG     000010    # EXEDF   # LOWCR
L.NAM     000002    # EXEDF   # LOWCR
L.TYPE    000005    # EXEDF   # LOWCR
L.UCB     000006    # EXEDF   # LOWCR
L.UNIT    000004    # EXEDF   # LOWCR
M$$MGE    000000    # LOWCR
NB.SNM    000040      ENTNM     FDRMV     LOCAT
NB.STP    000020      ENTNM     FDRMV     LOCAT
NB.SVR    000010      ENTNM     FDRMV     LOCAT
N.DID     000024      DRINI     FNDNM
N.FNAM    000006      FNDNM
N.FVER    000016      ENTNM     FDRMV
N.NEXT    000022      ENTNM     FDRMV
N.STAT    000020      ENTNM     FDRMV
P.ATT     000036    # LOWCR
P.BLKS    000016    # EXEDF   # LOWCR
P.BUSY    000024    # EXEDF   # LOWCR
P.HDR     000032      CRFIL   # LOWCR     PROCK
P.IOC     000003    # EXEDF   # LOWCR
P.LGTH    000042    # LOWCR
P.LNK     000000    # EXEDF   # LOWCR
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

P.MAIN    000012    # EXEDF   # LOWCR
P.NAM     000004    # EXEDF   # LOWCR
P.OWN     000026    # EXEDF   # LOWCR
P.PRI     000002    # EXEDF   # LOWCR
P.PRO     000034    # LOWCR
P.REL     000014    # EXEDF   # LOWCR
P.SIZE    000016    # EXEDF   # LOWCR
P.STAT    000030    # EXEDF   # LOWCR
P.SUB     000010    # EXEDF   # LOWCR
P.SWSZ    000022    # EXEDF   # LOWCR
P.TCB     000026    # EXEDF   # LOWCR
P.WAIT    000020    # EXEDF   # LOWCR
R.FIX     000001      DRACC
SP.EIP    000001    # EXEDF   # LOWCR
SP.ENB    000002    # EXEDF   # LOWCR
S.BMSK    177776    # EXEDF   # LOWCR
S.BMSV    177774    # EXEDF   # LOWCR
S.CCB     000030    # LOWCR
S.CON     000010    # EXEDF   # LOWCR
S.CSR     000012    # EXEDF   # LOWCR
S.CTM     000006    # EXEDF   # LOWCR
S.DRFN    000032      DREX      DRINI   # LOCAT     RMVNM
S.DZCK    000030    # LOWCR
S.FRK     000016    # EXEDF   # LOWCR
S.ITM     000007    # EXEDF   # LOWCR
S.LHD     000000    # EXEDF   # LOWCR
S.MPR     000030    # LOWCR
S.NFEN    000020      DRACC     ENTNM     FDRMV
S.PKT     000014    # EXEDF   # LOWCR
S.PRI     000004    # EXEDF   # LOWCR
S.RCNT    177772    # EXEDF   # LOWCR
S.ROFF    177773    # EXEDF   # LOWCR
S.STS     000011    # EXEDF   # LOWCR
S.VCT     000005    # EXEDF   # LOWCR
T.EXT     000000    # LOWCR
T.LGTH    000070    # LOWCR
UC.ATT    000010    # EXEDF   # LOWCR
UC.KIL    000004    # EXEDF   # LOWCR
UC.LGH    000003    # EXEDF   # LOWCR
UC.NPR    000100    # EXEDF   # LOWCR
UC.PWF    000020    # EXEDF   # LOWCR
UC.QUE    000040    # EXEDF   # LOWCR
UISAR0    177640    # EXEDF   # LOWCR
UISDR0    177600    # EXEDF   # LOWCR
US.BSY    000200    # EXEDF   # LOWCR
US.FOR    000040    # EXEDF   # LOWCR
US.MDM    000020      DISPAT    DMOUNT  # EXEDF   # LOWCR
US.MNT    000100      DMOUNT  # EXEDF   # LOWCR
US.OFL    000001    # EXEDF   # LOWCR
US.PUB    000004    # EXEDF   # LOWCR
US.UMD    000010    # EXEDF   # LOWCR
U.ACP     000032    # EXEDF   # LOWCR
U.ATT     000022    # EXEDF   # LOWCR
U.BUF     000024    # EXEDF   # LOWCR
U.CNT     000030    # EXEDF   # LOWCR
U.CTL     000004    # EXEDF   # LOWCR
U.CW1     000010    # EXEDF   # LOWCR
U.CW2     000012    # EXEDF   # LOWCR
U.CW3     000014    # EXEDF   # LOWCR
U.OWN     177776    # EXEDF   # LOWCR
U.RED     000002    # EXEDF   # LOWCR
U.SCB     000020    # EXEDF   # LOWCR
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

U.STS     000005      DISPAT    DMOUNT  # EXEDF   # LOWCR
U.ST2     000007    # EXEDF   # LOWCR
U.UNIT    000006    # EXEDF   # LOWCR
U.VCB     000034      CLNUP     DISPAT    DMOUNT  # EXEDF   # LOWCR
V$$CTR    000410    # LOWCR
X.AST     000032    # LOWCR
X.DSI     000024    # LOWCR
X.FORK    000012    # LOWCR
X.ISR     000010    # LOWCR
X.JSR     000002    # LOWCR
X.LEN     000050    # LOWCR
X.LNK     000000    # LOWCR
X.PSW     000006    # LOWCR
X.REL     000022    # LOWCR
X.TCB     000026    # LOWCR
X.VEC     000044    # LOWCR
X.VPC     000046    # LOWCR
$ABCTK    014460    # LOWCR
$ABTIM    005414    # LOWCR
$ABTSK    014464    # LOWCR
$ACHCK    007242    # LOWCR
$ACHKB    007250    # LOWCR
$ACHKP    007206    # LOWCR
$ACHKW    007232    # LOWCR
$ACTHD    005634    # LOWCR
$ACTRM    015172    # LOWCR
$ACTTK    014652    # LOWCR
$ALCLK    006636    # LOWCR
$ALEB1    032634    # LOWCR
$ALEMB    032620    # LOWCR
$ALERR    122774-R  # OVERR
$ALOCB    006524      ALLOC   # LOWCR
$ALOC1    006566    # LOWCR
$ALPKT    006652    # LOWCR
$BILDS    014540    # LOWCR
$BLKCK    010174    # LOWCR
$BLKC1    010204    # LOWCR
$BLXIO    006212      BLXIO   # LOWCR
$BMSET    032730    # LOWCR
$BTMSK    005640    # LOWCR
$CEFI     007362    # LOWCR
$CEFN     007356    # LOWCR
$CFLPT    005522    # LOWCR
$CHKPT    016226    # LOWCR
$CKACC    033726    # LOWCR
$CKCNT    005604    # LOWCR
$CKCSR    005606    # LOWCR
$CKINT    017726    # LOWCR
$CKLDC    005610    # LOWCR
$CLINS    014162    # LOWCR
$CLKHD    005556    # LOWCR
$CLRMV    014264    # LOWCR
$COMEF    005570    # LOWCR
$COPT     005560    # LOWCR
$CRASH    001470    # LOWCR
$CRATT    034024    # LOWCR
$CRAVL    005532    # LOWCR
$CRPAS    012470    # LOWCR
$CRSBF    000730    # LOWCR
$CRSBN    001462    # LOWCR
$CRSCS    001466    # LOWCR
$CRSHT    001752    # LOWCR
```

CROSS-REFERENCES

```
Symbol    Value      Modules That Reference Symbol

$CRSST     001460     # LOWCR
$CRSUN     001756     # LOWCR
$CVRTM     007070     # LOWCR
$C5TA      005772     # LOWCR
$DASTT     015030     # LOWCR
$DB0       043366     # LOWCR
$DDIV      121164-R   # DARITH    SMALC     SMDEL
$DEACB     006672     # LOWCR     RLEAS
$DEAC1     006732     # LOWCR
$DECLK     006644     # LOWCR
$DEPKT     006666     # LOWCR
$DETRG     031502     # LOWCR
$DEVHD     005462     # LOWCR
$DEVTB     043122     # LOWCR
$DIRSV     002264     # LOWCR
$DIRXT     002514     # LOWCR
$DIV       012564       DATIM     DLHDR     INWIN     IXEXT   # LOWCR
                        SMDEL
$DK0       043630     # LOWCR
$DMUL      121126-R   # DARITH    SMALC
$DPLM1     021662     # LOWCR
$DPLM2     021666     # LOWCR
$DQLM1     026526     # LOWCR
$DQLM2     026536     # LOWCR
$DRABO     022140     # LOWCR
$DRASG     022164     # LOWCR
$DRATP     032020     # LOWCR
$DRATR     031166     # LOWCR
$DRATX     022410     # LOWCR
$DRCEF     032340     # LOWCR
$DRCMT     022556     # LOWCR
$DRCRR     030600     # LOWCR
$DRCRW     024064     # LOWCR
$DRCSR     022562     # LOWCR
$DRDAR     022576     # LOWCR
$DRDCP     022632     # LOWCR
$DRDSE     032350     # LOWCR
$DRDTR     031350     # LOWCR
$DREAR     022612     # LOWCR
$DRECP     022656     # LOWCR
$DREIF     004220     # LOWCR
$DRELW     024312     # LOWCR
$DREXP     022702     # LOWCR
$DREXT     004226       DMOUNT  # LOWCR
$DRFEX     026100     # LOWCR
$DRGCL     023372     # LOWCR
$DRGLI     023500     # LOWCR
$DRGMX     025504     # LOWCR
$DRGPP     023602     # LOWCR
$DRGSS     023704     # LOWCR
$DRGTK     023714     # LOWCR
$DRGTP     024030     # LOWCR
$DRLM1     021566     # LOWCR
$DRLM2     021606     # LOWCR
$DRMAP     024346     # LOWCR
$DRMKT     025730     # LOWCR
$DRPUT     026122     # LOWCR
$DRQIO     026410     # LOWCR
$DRQRQ     027516     # LOWCR     RWVB
$DRRAF     032364     # LOWCR
$DRRCV     026162     # LOWCR
$DRREC     030306     # LOWCR
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

$DRREQ    031714    # LOWCR
$DRRES    031746    # LOWCR
$DRRRA    026142    # LOWCR
$DRRRF    025232    # LOWCR
$DRRUN    025762    # LOWCR
$DRSDV    032532    # LOWCR
$DRSEF    032420    # LOWCR
$DRSND    030440    # LOWCR
$DRSPN    032006    # LOWCR
$DRSRF    024716    # LOWCR
$DRSTV    032540    # LOWCR
$DRUNM    024672    # LOWCR
$DRWFL    032450    # LOWCR
$DRWFS    032512    # LOWCR
$DRWSE    032434    # LOWCR
$DSW      000046      OVERR     RWVBL     RW1LB
$DS0      044064    # LOWCR
$DTOER    032754    # LOWCR
$DT0      044276    # LOWCR
$DVCER    033016    # LOWCR
$DVERR    033016    # LOWCR
$DVMSG    007446      DMOUNT  # LOWCR
$DX0      044472    # LOWCR
$DYPMN    005470    # LOWCR
$EMSST    017156    # LOWCR
$EMTRP    021512    # LOWCR
$ERRHD    005700    # LOWCR
$ERRLM    005704    # LOWCR
$ERRPT    005514    # LOWCR
$ERRSQ    005706    # LOWCR
$ERRSV    005710    # LOWCR
$ERRSZ    005712    # LOWCR
$EXRQF    016434    # LOWCR
$EXRQN    016452    # LOWCR
$EXRQP    016426    # LOWCR
$EXSIZ    005520    # LOWCR
$FINBF    003776    # LOWCR
$FLTRP    017212    # LOWCR
$FMASK    005552      CRFIL   # LOWCR     PROCK
$FNDSP    015750    # LOWCR
$FORK     002320    # LOWCR
$FORK0    002342    # LOWCR
$FORK1    002340    # LOWCR
$FPINT    017226    # LOWCR
$FRKHD    005546    # LOWCR
$GTBYT    006102    # LOWCR
$GTPKT    007510    # LOWCR
$GTWRD    006160    # LOWCR
$HEADR    005564      DISPAT  # LOWCR     MPVBN
$ICHKP    016160    # LOWCR
$ILINS    017304    # LOWCR
$INITL    052414    # LOWCR
$INTCT    005542    # LOWCR
$INTSE    002376    # LOWCR
$INTSV    002434    # LOWCR
$INTXT    002374    # LOWCR
$INTX1    002456    # LOWCR
$IOABM    005714    # LOWCR
$IOALT    010272    # LOWCR
$IODON    010274    # LOWCR
$IOFIN    010414      DISPAT  # LOWCR
$IOKIL    010666    # LOWCR
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

$IOTRP    017316    # LOWCR
$LCKPR    011552    # LOWCR
$LDPWF    014124    # LOWCR
$LDRPT    005464    # LOWCR
$LOADR    041214    # LOWCR
$LOADT    016422    # LOWCR
$LOGHD    005624    # LOWCR
$LP0      044614    # LOWCR
$LSTLK    005630    # LOWCR
$MAPTK    017036    # LOWCR
$MCRCB    005626    # LOWCR
$MCRPT    005466    # LOWCR
$MM0      045014    # LOWCR
$MPCSR    012734    # LOWCR
$MPLND    011126    # LOWCR
$MPLNE    011106    # LOWCR
$MPLUN    011070    # LOWCR
$MPPHY    011170    # LOWCR
$MPPKT    011240    # LOWCR     RWVB
$MPVBN    011372    # LOWCR     MPVBN     RWVBL
$MUL      012534      DATIM   # LOWCR     RATCM     WATCM
$MXEXT    005770    # LOWCR
$NL0      051622    # LOWCR
$NONSI    002556    # LOWCR
$NS0      033300    # LOWCR
$NS1      033306    # LOWCR
$NS2      033314    # LOWCR
$NS3      033322    # LOWCR
$NS4      033330    # LOWCR
$NS5      033336    # LOWCR
$NS6      033344    # LOWCR
$NS7      033352    # LOWCR
$NULL     043100    # LOWCR
$NXTSK    015322    # LOWCR
$PANIC    001470    # LOWCR
$PARHD    005422    # LOWCR
$PARPT    005554    # LOWCR
$PARTB    012670    # LOWCR
$PCBS     052164    # LOWCR
$PKAVL    005764    # LOWCR
$PKMAX    005767    # LOWCR
$PKNUM    005766    # LOWCR
$POOL     052310    # LOWCR
$POWER    013750    # LOWCR
$PP0      045144    # LOWCR
$PR0      045266    # LOWCR
$PTBYT    006132    # LOWCR
$PTWRD    006160    # LOWCR
$PWRFL    005416    # LOWCR
$QASTT    015062    # LOWCR
$QEMB     033516    # LOWCR
$QINSF    014332    # LOWCR
$QINSP    014340    # LOWCR
$QMCRL    014376    # LOWCR
$QRMVF    014406      DISPAT  # LOWCR
$QRMVT    014420    # LOWCR
$RELOC    012222      BLXIO   # LOWCR
$RELOM    012272    # LOWCR
$RELOP    012410    # LOWCR
$RLMCB    023434    # LOWCR
$RLPAR    015246    # LOWCR
$RLPR1    015310    # LOWCR
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

$RQSCH    005452    # LOWCR
$SAVNR    004144    # LOWCR     WTRN1
$SCDVT    012306    # LOWCR
$SCDV1    012312    # LOWCR
$SETCR    014704    # LOWCR
$SETF     014762    # LOWCR
$SETM     014766    # LOWCR
$SETRQ    014734    # LOWCR
$SETRT    014732    # LOWCR
$SGFLT    017336    # LOWCR
$SHFPT    005516    # LOWCR
$SIGFL    005420    # LOWCR
$SRATT    034124    # LOWCR
$SRNAM    033644    # LOWCR
$SRSTD    015132    # LOWCR
$SRWND    034162    # LOWCR
$STACK    000642    # LOWCR
$STD      052220    # LOWCR
$STKDP    005454    # LOWCR
$STPCT    015222      DISPAT  # LOWCR
$STPTK    015226    # LOWCR
$SWSTK    004164    # LOWCR
$SYBEG    053424    # LOWCR
$SYSID    005574    # LOWCR
$SYSIZ    005716    # LOWCR
$SYTOP    063424    # LOWCR
$SYUIC    005612    # LOWCR
$TKNPT    005600    # LOWCR
$TKPS     005744    # LOWCR
$TKTCB    005446      DISPAT    DMOUNT  # LOWCR
$TKWSE    032430    # LOWCR
$TRACE    017376    # LOWCR
$TRP04    017410    # LOWCR
$TRTRP    021446    # LOWCR
$TSKHD    005512    # LOWCR
$TSKRP    016504    # LOWCR
$TSKRQ    016502    # LOWCR
$TSKRT    016476    # LOWCR
$TSTCP    016052    # LOWCR
$TTNS     005762    # LOWCR
$TT0      045534    # LOWCR
$TT1      045564    # LOWCR
$TT10     047672    # LOWCR
$TT11     047722    # LOWCR
$TT12     047752    # LOWCR
$TT13     050002    # LOWCR
$TT14     050032    # LOWCR
$TT15     050062    # LOWCR
$TT16     050112    # LOWCR
$TT17     050142    # LOWCR
$TT2      047452    # LOWCR
$TT20     050172    # LOWCR
$TT21     050222    # LOWCR
$TT22     051210    # LOWCR
$TT23     051260    # LOWCR
$TT24     051310    # LOWCR
$TT25     051340    # LOWCR
$TT26     051370    # LOWCR
$TT27     051420    # LOWCR
$TT3      047502    # LOWCR
$TT30     051450    # LOWCR
$TT31     051500    # LOWCR
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

$TT4      047532    # LOWCR
$TT5      047562    # LOWCR
$TT6      047612    # LOWCR
$TT7      047642    # LOWCR
$UISET    016776    # LOWCR
$UNMAP    034222    # LOWCR
$USRTB    000000    # LOWCR
$XDT      035002    # LOWCR
.ACBMX    000006    # ATCTL     RATCM     WATCM
.ACCES    136540-R  * ACCESS  @ DISPAT
.AGAIN    121406-R  # DISPAT    DREXT     IXEXT
.ALCAD    121640-R    EXCMP     EXCOM     EXTEN   # F11CM     SMALC
.ALCTL    121632-R    EXCMP     EXCOM     EXTEN   # F11CM     SMALC
.ALFCB    120252-R  # ALLOC     INFCB
.ALLOC    120274-R    ACCESS  # ALLOC     INWIN
.ALOBT    126144-R    SMALC   # SMSCN
.ATCTL    137414-R  * ATCTL     RATCM     WATCM
.ATMAX    000016    # ATCTL     RATCM     WATCM
.BLXI     120340-R  # BLXIO     DRINI     EXTEN     GTFID     WATCM
.BLXI1    120344-R  # BLXIO
.BLXO     120372-R  # BLXIO     CLCRE     CRFIL     DREX      EXCMP
                      EXCOM     RATCM
.BLXO1    120376-R  # BLXIO
.CKFRE    126136-R    SMALC   # SMSCN
.CKSM1    120430-R  # CKSUM
.CKSUM    120424-R  # CKSUM     DREXT     RDHDR     WRHDR
.CLACC    120450-R  # CLACC     CLNUP
.CLAC1    120516-R  # CLACC     CLDAC
.CLCRE    136722-R  * CLCRE     CLNUP
.CLDAC    121034-R  # CLDAC     CLNUP     DEACC
.CLDEL    136520-R  * CLCOM     CLNUP
.CLDIR    137030-R  * CLDIR     CLNUP     DREOF     DREXT     DRWRT
                      FNDNM
.CLEXI    137730-R  * CLNUP   @ DISPAT
.CLEXT    137134-R    CLCRE   * CLEXT     CLNUP   @ DISPAT
.CLEX1    121302-R  # DISPAT    TRUNC
.CLEX2    121304-R  # DISPAT    EXTHD
.CLFCB    137526-R  * CLFCB   @ DISPAT
.CLFC1    121316-R    CLCOM     CLDAC     CLEXT   # DISPAT
.CLNUP    137744-R  * CLNUP   @ F11CM
.CLRAT    136660-R  * CLCOM     CLNUP
.CLWAT    136702-R  * CLCOM     CLNUP
.CL0      051756    # LOWCR
.CO0      051732    # LOWCR
.CRFCB    122062-R    ACCESS    DRACC     EXTHD   # INFCB     RDATT
.CRFID    137466-R  * CRFID     CRFIL     EXTHD
.CRFIL    137670-R  * CRFIL   @ DISPAT
.DATIM    140346-R    CRFIL   * DATIM     DEACC
.DB0      043162    # LOWCR
.DB1      043222    # LOWCR
.DB2      043262    # LOWCR
.DB3      043322    # LOWCR
.DEACC    140502-R  * DEACC   @ DISPAT
.DELBT    126154-R    SMDEL   # SMSCN
.DK0      043464    # LOWCR
.DK1      043524    # LOWCR
.DK2      043564    # LOWCR
.DLBLK    140110-R    CLEXT   * DLBLK     DLFIL     DREXT
.DLBL1    140130-R    CLEXT   * DLBLK
.DLFIL    140304-R    CLCRE   @ DISPAT  * DLFIL
.DLFL1    121314-R    CLDAC   # DISPAT    DLMRK
.DLHDR    140356-R    CLCOM     CLEXT     DLFIL   * DLHDR
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

.DLHD1    140404-R    CLCRE     CLEXT   * DLHDR
.DLMRK    140516-R  @ DISPAT  * DLMRK
.DMOUN    140716-R  @ DISPAT  * DMOUNT
.DRACC    141160-R  @ DISPAT  * DRACC
.DRAC1    121324-R  # DISPAT    ENTNM     FNDNM     RMVNM
.DRALC    140652-R  * DRALC     DREXT
.DRBUF    131232-R    DRGET     DRWRT   # F11CM
.DRCPY    141704-R  * DRCPY     DREXT
.DREF1    121322-R  # DISPAT    DRWRT
.DRENB    134232-R    DRGET   # F11CM
.DREOF    142020-R  @ DISPAT  * DREOF
.DREX     142100-R  * DREX      FNDNM
.DREXT    142132-R  @ DISPAT  * DREXT
.DREX1    121320-R  # DISPAT    ENTNM
.DRFNB    122252-R    DRINI     ENTNM     FDRMV   # LOCAT
.DRFRE    122246-R    ENTNM   # LOCAT
.DRGET    142624-R  * DRGET     ENTNM   @ LOCAT
.DRHRC    122242-R  # LOCAT
.DRHVR    122244-R    ENTNM   # LOCAT
.DRINI    143050-R  * DRINI     ENTNM     FNDNM     RMVNM
.DRLBN    131226-R    DRWRT   # F11CM
.DRLVB    122250-R    ENTNM     FDRMV   # LOCAT
.DRNLB    134232-R    DRGET     ENTNM   # F11CM
.DRPAC    143172-R  * DRPAC     ENTNM     FNDNM     RMVNM
.DRSEF    142040-R  * DREOF     DREXT
.DRUCB    131224-R    CLNUP     DMOUNT    DRGET     ENTNM   # F11CM
                      SCFAC
.DRVLB    143250-R    DRCPY     DRGET   * DRVLB     ENTNM
.DRWEX    143416-R  * DRWRT     ENTNM     RMVNM
.DRWRT    143354-R  * DRWRT     ENTNM     RMVNM
.DRX1     142104-R  * DREX      DRWRT
.DSPAT    121452-R  # DISPAT    DRACC     DREXT
.DSW      000044    # LOWCR
.DS0      043720    # LOWCR
.DS1      043760    # LOWCR
.DS2      044020    # LOWCR
.DT0      044162    # LOWCR
.DT1      044226    # LOWCR
.DX0      044366    # LOWCR
.DX1      044426    # LOWCR
.ENTNM    143446-R  @ DISPAT  * ENTNM
.ENTRY    121330-R  # DISPAT
.ERMSG    121604-R  # F11CM
.EXCMP    141550-R  @ DISPAT  * EXCMP
.EXCM1    121276-R  # DISPAT    DREXT     EXTEN     IXEXT
.EXCM2    121310-R  # DISPAT    EXCOM
.EXCM3    121312-R  # DISPAT    EXCMP
.EXCNT    142624-R  @ DISPAT  * EXCOM
.EXCOM    142114-R  @ DISPAT  * EXCOM
.EXDSP    121616-R    CLCRE     CLNUP     DISPAT    DLFIL     DREXT
                    # F11CM     IXEXT
.EXFCB    121630-R    EXCMP     EXCOM     EXTHD   # F11CM
.EXFNU    121626-R    CLEXT     EXTHD   # F11CM
.EXHDJ    142030-R  @ DISPAT  * EXCMP
.EXHDR    121622-R    CLCOM     CLEXT     EXCOM   # F11CM     TRUNC
.EXIT     121510-R    CLNUP   # DISPAT
.EXNHD    121624-R    CLEXT     EXTHD   # F11CM     TRUNC
.EXSTS    121617-R    ACCESS    CLCRE     CLDAC     CLEXT     CLNUP
                      CRFIL     DEACC     EXCOM     EXTHD   # F11CM
                      RDATT     TRUNC     WRATT
.EXTEN    142770-R  @ DISPAT  * EXTEN
.EXTE1    143064-R  @ DISPAT  * EXTEN
```

```
Symbol  Value        Modules That Reference Symbol

.EXTHD     143320-R     EXCMP      * EXTHD
.EXTH1     121306-R   # DISPAT       EXCOM
.EXTN1     121274-R     CRFIL      # DISPAT
.EXTSV     121620-R     CLCOM        CLEXT        EXCOM      # F11CM        TRUNC
.FCBAD     121614-R     ACCESS       CLACC        CLCOM        CLDAC        CLDIR
                        CLFCB        DEACC        DLMRK        DRACC        DRCPY
                        DREOF        DREXT        DRGET        DRPAC        DRWRT
                        ENTNM        EXCMP        EXCOM        EXTHD        FDRMV
                        FNDNM      # F11CM        GTFID        INWIN        IXEXT
                        RATCM        RDATT        RDHDR        SCFAC        TRUNC
                        WACCK        WATCM
.FDRMV     144032-R   * FDRMV        FNDNM        RMVNM
.FILNO     121606-R     ACCESS       CLACC        CLCRE        CRFIL        DLMRK
                        DRINI        EXCMP        FNDNM      # F11CM        GTFID
                        IXEXT        RDATT        RDHDR        SCFCB
.FILSQ     121610-R     CRFIL        DRINI        FNDNM      # F11CM        IXEXT
                        RDHDR        SCFCB
.FNDNM     144360-R   @ DISPAT     * FNDNM
.FREPT     125620-R     EXCMP        EXCOM      # SMCOM
.FRLH      121644-R     ALLOC      # F11CM        INIT         RLEAS
.F1END     136134-R   # F11CM        INIT
.F1ORG     134234-R   # F11CM        INIT         RLEAS
.GTFID     121720-R     ACCESS       DEACC        DLMRK        EXTEN      # GTFID
                        RWATT
.GTMAP     122044-R     CLCOM        CLEXT        CLFCB        CRFIL        DLFIL
                        EXCMP        EXCOM        EXTHD      # GTMAP        INFCB
                        NXHDR        RDHDR        TRUNC
.HDBUF     130224-R     ACCESS       CLCOM        CLDAC        CLEXT        CRFIL
                        DEACC        DLBLK        DLHDR        DLMRK        DRACC
                        DREOF        DREXT        EXCMP        EXCOM        EXTHD
                      # F11CM        GTMAP        INFCB        NXHDR        PROCK
                        RATCM        RDHDR        TRUNC        WATCM        WRHDR
                        WTRN1
.HDLBN     130220-R     CRFIL        EXTHD      # F11CM        INFCB
.HDUCB     130216-R     CLCOM        CLNUP        CRFID        DLBLK        DLHDR
                        DMOUNT       DREXT        EXTHD      # F11CM        RDHDR
                        WRHDR
.INFCB     122100-R     CLFCB        DRACC        DREXT        EXCMP        EXTHD
                      # INFCB        RDATT        WATCM
.INIT      143772-R   @ DISPAT     * INIT
.INWIN     144034-R     ACCESS       DRACC      * INWIN
.IOPKT     121570-R     CLNUP        DISPAT       DLBLK        DRACC        DREX
                        DREXT        DRVLB        ENTNM        FDRMV      # F11CM
                        SMALC        SMNXB        TRUNC
.IOSTS     121600-R     ACCESS       CLEXT        CLNUP        DISPAT       DREXT
                        EXCMP        EXCOM        EXTHD      # F11CM        RWVBL
                        SMALC        WATCM
.IXEXT     144526-R   @ DISPAT     * IXEXT
.IXEX1     121300-R     CLNUP        CRFIL      # DISPAT
.LB0       052002     # LOWCR
.LDR       052220     # LOWCR
.LDRHD     052040     # LOWCR
.LOCAT     122304-R     ENTNM        FDRMV      # LOCAT
.LP0       044562     # LOWCR
.MM0       044704     # LOWCR
.MM1       044746     # LOWCR
.MPHDR     122604-R     CRFIL        EXTHD      # MPHDR        RDHDR
.MPVBN     122650-R     DRVLB        EXTEN        MPHDR      # MPVBN
.NDRLB     142126-R   * DREXT
.NDRSZ     142124-R   * DREXT
.NL0       051570     # LOWCR
.NOOP      121566-R   # DISPAT
```

CROSS-REFERENCES

```
Symbol  Value       Modules That Reference Symbol

.NXHDR    122706-R    ACCESS    EXCOM   # NXHDR     RDATT     TRUNC
.NXHD1    122724-R    CLEXT     DLFIL   # NXHDR
.PP0      045112    # LOWCR
.PRCK1    123024-R  # PROCK
.PRCK2    123034-R    DLMRK   # PROCK
.PRCK3    123044-R    ENTNM     FDRMV   # PROCK
.PROCK    123014-R    ACCESS    EXTEN   # PROCK     RWATT     TRUNC
                      WATCM
.PR0      045234    # LOWCR
.QIOST    121574-R  # F11CM     RWVBL     RW1LB
.RATCM    144370-R    ACCESS  * RATCM     RDATT
.RDATT    144614-R  @ DISPAT  * RDATT
.RDFHD    123256-R    ACCESS    CLCRE     CLFCB     DEACC     DLMRK
                      DRACC     DREOF     DREXT     EXCMP     EXTEN
                    # RDHDR     RWATT
.RDHDR    123316-R    CLFCB   # RDHDR     WITRN
.RDNLB    124222-R    DRGET   # RW1LB
.RD1LB    124242-R    CRFID     CRFIL     DLHDR     DRCPY     EXTHD
                      RDHDR   # RW1LB     SMRVB
.RHDFN    123334-R    CLEXT     EXTHD     NXHDR     RDATT   # RDHDR
.RHDLB    123342-R  # RDHDR
.RLEAS    123512-R    CLACC     CLCOM     CLDIR     CLNUP     DEACC
                      DMOUNT    EXTEN     RATCM     RDATT   # RLEAS
                      RLFCB     SCFAC     WATCM
.RLFCB    123554-R    CLACC     CLFCB   # RLFCB
.RMVNM    145002-R  @ DISPAT  * RMVNM
.RWATT    144772-R    RDATT   * RWATT     WRATT
.RWSIZ    124220-R  # RW1LB
.RWVB.    145046-R  @ DISPAT  * RWVB
.RWVBL    123646-R    DISPAT  # RWVBL
.RWVB1    121326-R  # DISPAT    RWVB
.SCFAC    124376-R    ACCESS    DLMRK     DRACC     DREXT     EXTEN
                      RWATT   # SCFAC
.SCFCB    124466-R    DRPAC     SCFAC   # SCFCB
.SMALC    124530-R    DRALC     EXCOM   # SMALC
.SMBUF    127216-R  # F11CM     INWIN     SMALC     SMDEL     SMNXB
.SMCNT    125614-R    SMALC   # SMCOM     SMDEL     SMNXB
.SMCTL    125611-R    DRALC     EXCMP     EXCOM     SMALC   # SMCOM
.SMDEL    145050-R    CLEXT     DLBLK   * SMDEL
.SMEXT    125610-R    EXCOM   # SMCOM
.SMFLG    100000    # F11CM     SMALC     SMNXB     SMRVB
.SMNXB    125622-R    SMALC   # SMNXB     SMSCN
.SMRVB    125730-R    CLEXT     DLFIL     DREXT     EXCMP     SMALC
                      SMDEL     SMNXB   # SMRVB
.SMSCN    126102-R    SMALC     SMDEL   # SMSCN
.SMUCB    127212-R    CLACC     CLCOM     CLNUP     DMOUNT  # F11CM
                      INWIN     SMALC     SMRVB
.SMVBN    127214-R    CLCOM     CLNUP   # F11CM     SMALC     SMNXB
                      SMSCN
.SM1AD    125572-R    SMALC   # SMCOM
.SM1BT    125564-R    CLEXT     DRALC     EXCMP     EXCOM     EXTHD
                      SMALC   # SMCOM     SMDEL     TRUNC
.SM1MK    125570-R    SMALC   # SMCOM
.SM1VB    125574-R    SMALC   # SMCOM     SMDEL
.SM2AD    125600-R  # SMCOM
.SM2BT    125604-R    SMALC   # SMCOM
.SM2MK    125602-R  # SMCOM
.SM2VB    125576-R    SMALC   # SMCOM
.SSTSZ    000007    # F11CM
.SSTVC    121646-R  # F11CM
.STACK    120252-R    CLNUP     DISPAT    DRACC   # F11CM
.START    121334-R  # DISPAT    RWVB
```

CROSS-REFERENCES

```
Symbol   Value       Modules That Reference Symbol

.SVLBN   125560-R    DLBLK     EXCMP     EXCOM   # SMCOM
.SY0     052026    # LOWCR
.TI0     051706    # LOWCR
.TRUNC   145274-R    DLMRK   * TRUNC
.TT0     045362    # LOWCR
.TT1     045452    # LOWCR
.TT10    046400    # LOWCR
.TT11    046470    # LOWCR
.TT12    046560    # LOWCR
.TT13    046650    # LOWCR
.TT14    046740    # LOWCR
.TT15    047030    # LOWCR
.TT16    047120    # LOWCR
.TT17    047210    # LOWCR
.TT2     045660    # LOWCR
.TT20    047300    # LOWCR
.TT21    047370    # LOWCR
.TT22    050316    # LOWCR
.TT23    050406    # LOWCR
.TT24    050476    # LOWCR
.TT25    050566    # LOWCR
.TT26    050656    # LOWCR
.TT27    050746    # LOWCR
.TT3     045750    # LOWCR
.TT30    051036    # LOWCR
.TT31    051126    # LOWCR
.TT4     046040    # LOWCR
.TT5     046130    # LOWCR
.TT6     046220    # LOWCR
.TT7     046310    # LOWCR
.UCBAD   121572-R    CLACC     CLNUP     CRFID     CRFIL     DISPAT
                     DLBLK     DMOUNT    DREXT     DRGET     ENTNM
                   # F11CM     PROCK     RDHDR     RWVB      SCFAC
                     SMALC     SMRVB     WRHDR
.USEPT   125616-R    EXCMP     EXCOM   # SMCOM
.WACCK   145172-R    DRACC     EXTEN   * WACCK     WRATT
.WATCM   145212-R    CRFIL     DEACC   * WATCM     WRATT
.WITRN   126172-R    DRVLB     EXTEN     INWIN     IXEXT     MPHDR
                     RWVB      RWVBL   # WITRN
.WNDOW   121612-R    ACCESS    CLACC     CLDIR     DEACC     DRVLB
                     EXCMP     EXTEN     EXTHD   # F11CM     GTFID
                     INWIN     IXEXT     TRUNC     WRATT
.WRATT   145542-R  @ DISPAT  * WRATT
.WRHDR   126272-R    CLCOM     CLDAC     CLEXT     CRFIL     DLMRK
                     DREOF     DREXT     EXCMP     EXTHD     WRATT
                   # WRHDR
.WRHD1   126304-R    CRFID     DLHDR   # WRHDR
.WR1LB   124234-R    DLHDR     DRCPY     DRWRT   # RW1LB     SMRVB
                     WRHDR
.WTRN1   126330-R    ACCESS    EXCMP     EXTHD     WITRN   # WTRN1
.ZERCT   000021      DISPAT  # F11CM
```

9.8 BIGFCP SEGMENT CROSS-REFERENCES

The BIGFCP segment cross-reference lists the name of each overlay and the modules that compose it. The cross-reference follows:

Segment Name	Resident Modules					
FCPHI	ACCESS	ATCTL	CRFID	CRFIL	DATIM	DEACC
	DMOUNT	DRACC	EXCMP	EXCOM	EXTEN	EXTHD
	INIT	INWIN	RATCM	RDATT	RWATT	RWVB
	WACCK	WATCM	WRATT			
FCPLO	CLCOM	CLCRE	CLDIR	CLEXT	CLFCB	CLNUP
	DLBLK	DLFIL	DLHDR	DLMRK	DRALC	DRCPY
	DREOF	DREX	DREXT	DRGET	DRINI	DRPAC
	DRVLB	DRWRT	ENTNM	FDRMV	FNDNM	IXEXT
	RMVNM	SMDEL	TRUNC			
F11ACP	ALLOC	BLXIO	CKSUM	CLACC	CLDAC	DARITH
	DISPAT	EXEDF	F11ACP	F11CM	GTFID	GTMAP
	INFCB	LOCAT	LOWCR	MPHDR	MPVBN	NXHDR
	OVERR	PROCK	RDHDR	RLEAS	RLFCB	RWVBL
	RW1LB	SCFAC	SCFCB	SMALC	SMCOM	SMNXB
	SMRVB	SMSCN	WITRN	WRHDR	WTRN1	

9.9 CONDITIONAL ASSEMBLY PARAMETER TO MODULE CROSS-REFERENCE

This cross-reference contains a listing of the conditional assembly parameters that are contained in the Executive modules. Listed to the right of each parameter are those Executive modules that contain conditional assemblies affected by the parameter.

Conditional Assembly Parameter	Modules That Contain The Parameter					
A$$CHK	DRATX	DRDSP	DRGLI	DRGPP	DRGTK	DRGTP
	DRQIO	DRRAS	DRSED	DRSST	IOSUB	SSTSR
	TTDRV					
A$$CPS	DKTAB	DMTAB	DRQIO	DRRES	DTTAB	DXTAB
	IOSUB					
A$$D01	BFCTL					
A$$F11	BFCTL					
A$$NSI	DRQIO	MMTAB	MTDRV	MTTAB		
A$$PRI	DRDSP	DREIF	DRRES			
A$$TRP	DRATX	DRCIN	DRDAR	DRDSP	DREIF	DREXP
	DRMAP	DRPUT	DRREG	IOSUB	LOADR	POWER
	REQSB	SYSXT	TDSCH	TTDRV		
B$$OOT	CRASH					
C$$CDA	CRASH					

Conditional Assembly Parameter	Modules That Contain The Parameter					
C$$CKP	DRATX	DRDCP	DRDSP	DREIF	DREXP	DRREG
	IOSUB	LOADR	REQSB	SYSCM	SYSXT	TDSCH
	TTDRV					
C$$INT	DRCIN	DRDCP	REQSB	SYSXT	DRDSP	DREIF
C$$MPT	XUDRV					
C$$RSH	CRASH					
D$$B11	XBDRV					
D$$B11-1	XBDRV					
D$$H11	LOWCR	TTDRV				
D$$IAG	CTTAB	DBDRV	DKTAB	DLDRV	DMDRV	DMTAB
	DPDRV	DRQIO	DTDRV	DTTAB	DXTAB	ERROR
	IOSUB	LPTAB	MMDRV	MMTAB	MTDRV	MTTAB
	PPTAB	SYSTB				
D$$ISK	DRATX	DRDCP	DRDSP	DREIF	DRREG	DRRES
	IOSUB	LOADR	REQSB	TDSCH		
D$$J11	TTDRV					
D$$L11	TTDRV					
D$$M11	TTDRV					
D$$P11	XPDRV					
D$$P11-1	XPDRV					
D$$Q11	XQDRV					
D$$Q11-1	XQDRV					
D$$SHF	DRCIN	IOSUB	PARTY	REQSB	TDSCH	
D$$U11	XUDRV					
D$$U11-1	XUDRV					
D$$W11	XWDRV					
D$$W11-1	XWDRV					
D$$WCK	DBDRV	DMDRV	DPDRV			
D$$YNC	DREXP	LOADR	REQSB	SYSCM		
D$$YNM	DRCIN	DRDSP	DREIF	DREXP	DRGPP	DRREG
	DRREQ	IOSUB	PLSUB	REQSB	SYSCM	SYSDF
	SYSXT	TDSCH	TTDRV			
D$$Z11	TTDRV					
D$$ZMD	TTDRV					

CROSS-REFERENCES

Conditional Assembly Parameter	Modules That Contain The Parameter					
E$$DVC	CTDRV	CTTAB	DBDRV	DKTAB	DLDRV	DMDRV
	DMTAB	DPDRV	DRDRV	DTDRV	DTTAB	DXDRV
	DXTAB	ERROR	IOSUB	LPTAB	MMDRV	MMTAB
	MTDRV	MTTAB	POWER	PPTAB	SYSCM	SYSTB
	SYSXT					
E$$EAE	POWER	SYSXT				
E$$NSI	DRCIN	ERROR	LOWCR	POWER	SYSCM	SYSXT
E$$PER	ERROR	PARTY	POWER			
E$$XPR	DRDSP	DREXP	LOADR	SYSCM		
F$$AST	POWER	SSTSR				
F$$LPP	DRDSP	DREIF	DRPUT	POWER	REQSB	SSTSR
	SYSXT					
F$$LTP	SSTSR					
G$$TPP	DRDSP	DRGPP				
G$$TSS	DRDSP	DRGSS				
G$$TTK	DRDSP	DRGTK				
G$$WRD	BFCTL					
I$$C11	LOWCR					
I$$CAD	BFCTL					
I$$RAR	DREIF	TTDRV				
I$$RDN	DREIF	TTDRV				
K$$CNT	SYSCM					
K$$CSR	SYSCM					
K$$LDC	SYSCM					
K$$W11	POWER	TDSCH				
L$$11R	LPDRV					
L$$50H	TTDRV					
L$$ASG	DRASG					
L$$DRV	CTTAB	DKTAB	DMTAB	DRGLI	DRQIO	DTTAB
	DXTAB	IOSUB	LPTAB	MMTAB	MTTAB	POWER
	PPTAB	QUEUE	REQSB	SYSCM	SYSTB	SYSXT
	TDSCH	TTDRV				
L$$LDR	LOADR	SYSCM	SYTAB			
L$$P11	LPDRV					

CROSS-REFERENCES

Conditional Assembly Parameter	Modules That Contain The Parameter					
L$$SI1	CRASH	INITL	POWER	SYSCM	SYSXT	TTDRV
LD$$H	TTDRV					
LD$$J	TTDRV					
LD$$L	TTDRV					
LD$$Z	TTDRV					
LD$CT	CTTAB					
LD$DK	DKTAB					
LD$DM	DMTAB					
LD$DT	DTTAB					
LD$DX	DXTAB					
LD$LP	LPTAB					
LD$MM	MMTAB					
LD$MT	MTTAB					
LD$NL	SYSTB					
LD$PP	PPTAB					
LD$PR	PPTAB					
LD$TT	SYSTB	TTDRV				
LD$TT	XMDRV					
M$$CRI	TTDRV					
M$$CRX	DRDSP	DREIF	DRGCL			
M$$EXT	DBDRV	DKTAB	DLDRV	DMDRV	DMTAB	DPDRV
	DRDRV	DTDRV	DTTAB	INITL	IOSUB	MMDRV
	MTDRV	MTTAB	POWER	SYSCM	SYSDF	XBDRV
	XMDRV	XQDRV				
M$$IXD	DBDRV	IOSUB	MMDRV			
M$$MGE	BFCTL	CRASH	CTTAB	DBDRV	DKTAB	DLDRV
	DMDRV	DMTAB	DPDRV	DRATX	DRCIN	DRDRV
	DRDSP	DREIF	DREXP	DRGLI	DRGPP	DRGTK
	DRGTP	DRQIO	DRRAS	DRREG	DRREQ	DRSED
	DRSST	DTDRV	DTTAB	DXDRV	DXTAB	INITL
	IOSUB	LOADR	LOWCR	LPDRV	LPTAB	MMDRV
	MMTAB	MTDRV	MTTAB	PARTY	PLSUB	POWER
	PPTAB	QUEUE	REQSB	SSTSR	SYSCM	SYSDF
	SYSTB	SYSXT	SYTAB	TDSCH	TTDRV	XBDRV
	XMDRV	XPDRV	XQDRV	XUDRV	XWDRV	

Conditional Assembly Parameter	Modules That Contain The Parameter
M$$MUP	CTTAB DKTAB DMTAB DRASG DRDSP DRGTK DRMKT DRQIO DRRAS DRREQ DRRES DTTAB DXTAB LPTAB MMTAB MTTAB PPTAB REQSB SYSCM SYSTB TTDRV
M$$NET	DRQIO LOWCR XBDRV XMDRV XPDRV XQDRV XUDRV XWDRV
N$$MOV	BFCTL
N$$UMR	SYSCM
P$$D70	PARTY
P$$GMX	DRDSP DRMAP
P$$LAS	DRDSP DREIF DREXP DRGPP DRMAP DRPUT DRREG LOADR LOWCR PLSUB REQSB SSTSR SYSCM SYSXT SYTAB TTDRV
P$$P45	NULTK
P$$R11	PRDRV
P$$RFL	DRDSP DREIF DRPUT POWER REQSB
P$$RTY	INITL PARTY POWER SYSCM
P$$SRF	DRDSP DREIF DRMAP DRPUT LOADR REQSB SYTAB
P$$WRD	BFCTL
Q$$22	XMDRV
Q$$IO	XMDRV
Q$$OPT	CORAL IOSUB SYSCM
Q$$CRC	XQDRV
Q$$MPT	XQDRV
R$$11S	DRGTK INITL LOADR SYSCM SYTAB TTDRV
R$$60F	DMDRV IOSUB
R$$611	DMDRV IOSUB
R$$DER	CORAL
R$$EXV	SYSCM SYSXT
R$$JP1	DBDRV IOSUB
R$$JPO	DBDRV IOSUB
R$$JS1	IOSUB

Conditional Assembly Parameter	Modules That Contain The Parameter					
R$$K11	IOSUB					
R$$L11	DLDRV	IOSUB				
R$$LKL	DRQIO	IOSUB	SYSXT	TTDRV		
R$$M11	IOSUB					
R$$MOF	IOSUB					
R$$NDC	TDSCH					
R$$P11	DPDRV	IOSUB				
R$$SND	DRDSP	DREIF	DRPUT	DRRAS	LOADR	REQSB
R$$X11	DXDRV					
S$$ECC	DBDRV	DMDRV	IOSUB			
S$$WPC	LOADR	REQSB	TDSCH			
S$$WPR	LOADR					
S$$YSZ	SYSCM					
T$$18S	TTDRV					
T$$30P	SYSXT	TTDRV				
T$$A11	CTDRV					
T$$ACR	SYSTB	TTDRV				
T$$BTW	SYSTB	TTDRV				
T$$BUF	DREIF	DREXP	DRREG	SYSXT	TTDRV	
T$$C11	DTDRV	IOSUB				
T$$CCA	SYSTB	TTDRV				
T$$CCO	SYSTB	TTDRV				
T$$CTR	TTDRV					
T$$ESC	SYSTB	TTDRV				
T$$GMC	SYSTB	TTDRV				
T$$GTS	SYSTB	TTDRV				
T$$HLD	SYSTB	TTDRV				
T$$J16	MMDRV					
T$$KMG	DREIF	IOSUB				
T$$LWC	TTDRV					

Conditional Assembly Parameter	Modules That Contain The Parameter	
T$$M11	MTDRV	
T$$MIN	TTDRV	
T$$RNE	TTDRV	
T$$RPR	SYSTB	TTDRV
T$$RST	TTDRV	
T$$RUB	TTDRV	
T$$SMC	SYSXT	TTDRV
T$$SYN	SYSXT	TTDRV
T$$TRW	SYSXT	TTDRV
T$$UTB	SYSXT	TTDRV
T$$VBF	SYSXT	TTDRV
U$$ADM	BFCTL	
V$$CTR	LOWCR	SYSDF
X$$18	XMDRV	
X$$22	XMDRV	
X$$LDM	XMDRV	
X$$M11-1	XMDRV	
X$$M11	XMDRV	

9.10 MODULE TO CONDITIONAL ASSEMBLY PARAMETER CROSS-REFERENCE

This cross-reference contains a listing of the Executive modules that contain conditional assembly parameters. Listed to the right of each module are the parameters that affect the assembly of the module.

Module	Conditional Assembly Parameters in Module				
BFCTL	A$$D01	A$$F11	G$$WRD	I$$CAD	M$$MGE
	N$$MOV	P$$WRD	U$$ADM		
CORAL	Q$$OPT	R$$DER			
CRASH	B$$OOT	C$$CDA	C$$RSH	L$$SI1	M$$MGE
CTDRV	E$$DVC	T$$A11			
CTTAB	D$$IAG	E$$DVC	L$$DRV	LD$CT	M$$MGE
	M$$MUP				

Module	Conditional Assembly Parameters in Module				
DBDRV	D$$IAG	D$$WCK	E$$DVC	M$$EXT	M$$IXD
	M$$MGE	R$$JP1	R$$JPO	S$$ECC	
DKTAB	A$$CPS	D$$IAG	E$$DVC	L$$DRV	LD$DK
	M$$EXT	M$$MUP			
DLDRV	D$$IAG	E$$DVC	M$$EXT	M$$MGE	R$$L11
DMDRV	D$$IAG	D$$WCK	E$$DVC	M$$EXT	M$$MGE
	R$$60F	R$$611	S$$ECC		
DMTAB	A$$CPS	D$$IAG	E$$DVC	L$$DRV	LD$DM
	M$$EXT	M$$MGE	M$$MUP		
DPDRV	D$$IAG	D$$WCK	E$$DVC	M$$EXT	M$$MGE
	R$$P11				
DRASG	L$$ASG	M$$MUP			
DRATX	A$$CHK	A$$TRP	C$$CKP	D$$ISK	M$$MGE
DRCIN	A$$TRP	C$$INT	D$$SHF	D$$YNM	E$$NSI
	M$$MGE				
DRDAR	A$$TRP				
DRDCP	C$$CKP	C$$INT	D$$ISK		
DRDRV	E$$DVC	M$$EXT	M$$MGE		
DRDSP	A$$CHK	A$$PRI	A$$TRP	C$$CKP	C$$INT
	D$$ISK	D$$YNM	E$$XPR	F$$LPP	G$$TPP
	G$$TSS	G$$TTK	M$$CRX	M$$MGE	M$$MUP
	P$$GMX	P$$LAS	P$$RFL	P$$SRF	R$$SND
DREIF	A$$PRI	A$$TRP	C$$CKP	C$$INT	D$$ISK
	D$$YNM	F$$LPP	I$$RAR	I$$RDN	M$$CRX
	M$$MGE	P$$LAS	P$$RFL	P$$SRF	R$$SND
	T$$BUF	T$$KMG			
DREXP	A$$TRP	C$$CKP	D$$YNC	D$$YNM	E$$XPR
	M$$MGE	P$$LAS	T$$BUF		
DRGCL	M$$CRX				
DRGLI	A$$CHK	L$$DRV	M$$MGE		
DRGPP	A$$CHK	D$$YNM	G$$TPP	M$$MGE	P$$LAS
DRGSS	G$$TSS				
DRGTK	A$$CHK	G$$TTK	M$$MGE	M$$MUP	R$$11S
DRGTP	A$$CHK	M$$MGE			
DRMAP	A$$TRP	P$$GMX	P$$LAS	P$$SRF	
DRMKT	M$$MUP				

Module	Conditional Assembly Parameters in Module				
DRPUT	A$$TRP	F$$LPP	P$$LAS	P$$RFL	P$$SRF
	R$$SND				
DRQIO	A$$CHK	A$$CPS	A$$NSI	D$$IAG	L$$DRV
	M$$MGE	M$$MUP	M$$NET	R$$LKL	
DRRAS	A$$CHK	M$$MGE	M$$MUP	R$$SND	
DRREG	A$$TRP	C$$CKP	D$$ISK	D$$YNM	M$$MGE
	P$$LAS	T$$BUF			
DRREQ	D$$YNM	M$$MGE	M$$MUP		
DRRES	A$$CPS	A$$PRI	D$$ISK	M$$MUP	
DRSED	A$$CHK	M$$MGE			
DRSST	A$$CHK	M$$MGE			
DTDRV	D$$IAG	E$$DVC	M$$EXT	M$$MGE	T$$C11
DTTAB	A$$CPS	D$$IAG	E$$DVC	L$$DRV	LD$DT
	M$$EXT	M$$MGE	M$$MUP		
DXDRV	E$$DVC	M$$MGE	R$$X11		
DXTAB	A$$CPS	D$$IAG	E$$DVC	L$$DRV	LD$DX
	M$$MGE	M$$MUP			
ERROR	D$$IAG	E$$DVC	E$$NSI	E$$PER	
INITL	L$$SI1	M$$EXT	M$$MGE	P$$RTY	R$$11S
IOSUB	A$$CHK	A$$CPS	A$$TRP	C$$CKP	D$$IAG
	D$$ISK	D$$SHF	D$$YNM	E$$DVC	L$$DRV
	M$$EXT	M$$IXD	M$$MGE	Q$$OPT	R$$60F
	R$$611	R$$JPO	R$$JS1	R$$C11	R$$K11
	R$$L11	R$$M11	R$$P11	R$$LKL	R$$MOF
	S$$ECC	T$$KMG			
LOADR	A$$TRP	C$$CKP	D$$ISK	D$$YNC	E$$XPR
	L$$LDR	M$$MGE	P$$LAS	P$$SRF	R$$11S
	R$$SND	S$$WPC	S$$WPR		
LOWCR	D$$H11	E$$NSI	I$$C11	M$$MGE	M$$NET
	P$$LAS	V$$CTR			
LPDRV	L$$11R	L$$P11	M$$MGE		
LPTAB	D$$IAG	E$$DVC	L$$DRV	LD$LP	M$$MGE
	M$$MUP				
MMDRV	D$$IAG	E$$DVC	M$$EXT	M$$IXD	M$$MGE
	T$$J16				
MMTAB	A$$NSI	D$$IAG	E$$DVC	L$$DRV	LD$MM
	LD$MT	M$$MGE	M$$MUP		
MTDRV	A$$NSI	D$$IAG	E$$DVC	M$$EXT	M$$MGE
	T$$M11				

CROSS-REFERENCES

Module	Conditional Assembly Parameters in Module				
MTTAB	A$$NSI	D$$IAG	E$$DVC	L$$DRV	LD$MT
	M$$EXT	M$$MGE	M$$MUP		
NULTK	P$$P45				
PARTY	D$$SHF	E$$PER	M$$MGE	P$$D70	P$$RTY
PLSUB	D$$YNM	M$$MGE	P$$LAS		
POWER	A$$TRP	E$$DVC	E$$EAE	E$$NSI	E$$PER
	F$$AST	F$$LPP	K$$W11	L$$DRV	L$$SI1
	M$$EXT	M$$MGE	P$$RFL	P$$RTY	
PPTAB	D$$IAG	E$$DVC	L$$DRV	LD$PP	LD$PR
	M$$MGE	M$$MUP			
PRDRV	P$$R11				
QUEUE	L$$DRV	M$$MGE			
REQSB	A$$TRP	C$$CKP	C$$INT	D$$ISK	D$$SHF
	D$$YNC	D$$YNM	F$$LPP	L$$DRV	M$$MGE
	M$$MUP	P$$LAS	P$$RFL	P$$SRF	R$$SND
	S$$WPC				
SSTSR	A$$CHK	F$$AST	F$$LPP	F$$LTP	M$$MGE
	P$$LAS				
SYSCM	C$$CKP	D$$YNC	D$$YNM	E$$DVC	E$$NSI
	E$$XPR	K$$CNT	K$$CSR	K$$LDC	L$$DRV
	L$$LDR	L$$SI1	M$$EXT	M$$MGE	M$$MUP
	N$$UMR	P$$LAS	P$$RTY	Q$$OPT	R$$11S
	R$$EXV	S$$YSZ			
SYSDF	D$$YNM	M$$EXT	M$$MGE	V$$CTR	
SYSTB	D$$IAG	E$$DVC	L$$DRV	LD$NL	LD$TT
	M$$MGE	M$$MUP	T$$ACR	T$$BTW	T$$CCA
	T$$CCO	T$$ESC	T$$GMC	T$$GTS	T$$HLD
	T$$RPR				
SYSXT	A$$TRP	C$$CKP	C$$INT	D$$YNM	E$$DVC
	E$$EAE	E$$NSI	F$$LPP	L$$DRV	L$$SI1
	M$$MGE	P$$LAS	R$$EXV	R$$LKL	T$$BUF
	T$$SMC	T$$SYN	T$$TRW	T$$UTB	T$$30P
	T$$VBF				
SYTAB	L$$LDR	M$$MGE	P$$LAS	P$$SRF	R$$11S
TDSCH	A$$TRP	C$$CKP	D$$ISK	D$$SHF	D$$YNM
	K$$W11	L$$DRV	M$$MGE	R$$NDC	S$$WPC

CROSS-REFERENCES

Module	Conditional Assembly Parameters in Module				
TTDRV	A$$CHK	A$$TRP	C$$CKP	D$$H11	D$$J11
	D$$L11	D$$M11	D$$YNM	D$$Z11	D$$ZMD
	I$$RAR	I$$RDN	L$$50H	L$$DRV	L$$SI1
	LD$$H	LD$$J	LD$$L	LD$$Z	LD$TT
	M$$CRI	M$$MGE	M$$MUP	P$$LAS	R$$11S
	R$$LKL	T$$18S	T$$30P	T$$ACR	T$$BTW
	T$$BUF	T$$CCA	T$$CCO	T$$CTR	T$$ESC
	T$$GMC	T$$GTS	T$$HLD	T$$LWC	T$$MIN
	T$$RNE	T$$RPR	T$$RST	T$$RUB	T$$SMC
	T$$SYN	T$$TRW	T$$UTB	T$$VBF	
XBDRV	D$$B11	M$$EXT	M$$MGE	M$$NET	
XMDRV	LD$XM	M$$EXT	M$$MGE	M$$NET	Q$$22
	Q$$IO	X$$18	X$$22	X$$LDM	X$$M11
	X$$M11-1				
XPDRV	M$$MGE	M$$NET	D$$P11-1	D$$P11	
XQDRV	D$$Q11-1	D$$Q11	M$$EXT	M$$MGE	M$$NET
	Q$$CRC	Q$$HPT			
XUDRV	C$$MPT	D$$U11-1	D$$U11	M$$MGE	M$$NET
XWDRV	C$$MPT	D$$W11-1	D$$W11	M$$MGE	M$$NET