RSX-11M Operating System Internals

Student Workbook

digital

RSX-11M Operating System Internals

Student Workbook

**The manuscript for this book was created using DIGITAL Standard
Runoff. Book production was done by Educational Services
Development and Publishing in South Lawrence, MA.**

The following are trademarks of Digital Equipment Corporation:

d i g i t a l	DECtape	Rainbow
DATATRIEVE	DECUS	RSTS
DEC	DECwriter	RSX
DECmate	DIBOL	UNIBUS
DECnet	MASSBUS	VAX
DECset	PDP	VMS
DECsystem-10	P/OS	VT
DECSYSTEM-20	Professional	Work Processor

CONTENTS

SG STUDENT GUIDE

1 OVERVIEW OF RSX-11M

2 THE SYSTEM DATA BASE

3 EXECUTIVE MODULES

4 FILE SYSTEM OVERVIEW

5 SYSTEM GENERATION

6 PHYSICAL AND VIRTUAL MEMORY

7 TASK MANAGEMENT

10 SYSTEM CRASHES

11 DIRECTIVE PROCESSING

12 I/O PROCESSING

13 DEVICE DRIVER PROCESSING

FIGURES

STUDENT GUIDE

TABLES

EXAMPLES

INTRODUCTION

This course is designed for the programmer of an RSX-11M system. A system programmer must troubleshoot system problems and write needed privileged code.

As a system programmer, your responsibilities will include the following:

- Determining features needed on system.

- Performing System Generations (SYSGENs).

- Maintaining the software by installing patches.

- Determining the cause(s) of system crashes.

- Providing needed privileged tasks.

- Writing new directive code.

- Assisting application programmers with system problems.

This course is designed to give you the knowledge and skills necessary to carry out these tasks.

COURSE GOALS

Upon completing this course, you should be able to:

1. Install and maintain RSX-11M software.

2. Use output from CDA to solve system problems.

3. Write privileged tasks which access the system data base.

4. Write privileged tasks which call system routines.

5. Write a new directive processing routine and add the directive to the system.

6. Determine system status by examining the system data base.

7. Use knowledge of the operating system to improve performance of applications software.

NON-GOALS

The following are not goals of this course:

1. The course does not teach you how to write a device driver.

2. The course does not teach you how to create a user-written Command Line Interpreter.

3. The course does not teach you how to make any modifications to system source code. Specifically, it will not teach you how to modify DCL in any way.

PREREQUISITES

You should have already taken RSX-11M ` Programming in FORTRAN/MACRO, or have equivalent experience using RSX-11M. You should also be able to read and understand MACRO-11 code.

COURSE ORGANIZATION

The student workbook handout is organized into modules which correspond to the major topics covered by the instructor during the course. Each module is designed to support the lecture by providing you with notes and exercises to guide your study.

A module is composed of five parts:

1. An Introduction which provides an overview of the module. It explains in general terms what the module covers and how the material within the module is related to your needs. Often, it introduces some of the important concepts detailed in the module.

2. Objectives which list the knowledge and skills you should obtain as a result of studying the module. Read these before the corresponding lecture, and review them after completion of the module and its exercises.

3. Resources which provide a list of reference materials for further information about the topics of the module and related topics. You should familiarize yourself with the reference materials because they are your primary source of information after you complete the course.

   Your instructor may suggest selected readings as preparation for a lecture, or for review and study after the lecture.

4. The module text which provides brief notes on topics covered in the lecture, with printed copies of lists, tables, diagrams, etc. your instructor may use in class. The texts provide you with notes to review after the lecture, and reduces somewhat the note-taking you must do during lecture.

    Note that the module text is a supplement to the lecture. It is not intended to be tutorial reading for learning the material on your own. Nor is it intended to replace the function of reference manuals which fully document the software. You are strongly encouraged to supplement these notes with notes of your own and with references to the system documentation.

5. Module Exercises which are designed to help you practice and test the knowledge acquired during lecture and study. If you can perform all of the written and/or lab exercises at the end of the module, you have mastered the objectives of that module.

    Your instructor may suggest particular lab exercises to concentrate on, if time or equipment limitations make it impossible to perform all of them. In addition, your instructor may provide additional suggestions for lab activities.

Modules

There are thirteen modules of this course. The following is a brief list of their major topics:

1. OVERVIEW OF RSX-11M describes the operating system and its goals.

2. THE SYSTEM DATA BASE describes the most important data structures and lists in the system data base.

3. EXECUTIVE MODULES describes the most important Executive modules.

4. FILE SYSTEM OVERVIEW describes the structure of a FILES-11 volume and the data structures needed to process it.

5. SYSTEM GENERATION describes the system generation process. (For reference only; this module is optional.)

6. PHYSICAL AND VIRTUAL MEMORY describes partitions, allocation of memory to tasks, checkpointing, function and use of memory management.

7. TASK MANAGEMENT describes the life cycle of a task.

8. SYSTEM SYNCHRONIZATION describes trap and interrupt processing, and access to the system data base.

9. WRITING PRIVILEGED TASKS describes writing, assembling, and building privileged tasks.

10. SYSTEM CRASHES uses CDA to troubleshoot system problems.

11. DIRECTIVE PROCESSING describes the directive dispatcher, the directive dispatch table, and directive processing routines.

12. I/O PROCESSING describes the components of the I/O system, ACPs, and QIO processing.

13. DEVICE DRIVER PROCESSING describes the driver data base and the structure of standard device drivers.

RESOURCES

1. RSX-11M Release Notes (AA-2573G-TC)

   Contains the most recent version information and should be read thoroughly before any system generation. Includes information on patching.

2. RSX-11M System Generation and Installation Guide

   Contains complete documentation on the installation and SYSGEN procedures.

3. RSX-11M/M-Plus I/O Operations Reference Manual

   Contains documentation on the FCS macros and subroutines.

4. RSX-11M/M-Plus Executive Reference Manual

   Contains descriptions of all system directives.

5. RSX-11M/M-Plus Crash Dump Analyzer

   Explains how to use the CDA facilities.

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

   Contains a detailed description of the procedures involved in writing a new device driver.

1

OVERVIEW OF RSX-11M

INTRODUCTION

RSX-11M is designed to support many applications with particular emphasis on real-time applications. For this reason, a large number of services are provided to the programmer. This module presents a brief overview of these features, many of which will be examined in detail in later modules.

OBJECTIVES

1. List the major design goals of RSX-11M.

2. List the major features of RSX-11M.

3. List the major components of RSX-11M.

DESIGN GOALS

- Real-time system with

    8K word minimum Executive

    16K word minimal system (disk based, run time only)

    28K word system required for SYSGEN (version 4.0)

- Upward compatible with RSX-11D

- Support

    All PDP-11 processors

    Most processor options

    Full line of peripherals

- Provide system and task synchronization not relying on task priorities

- Provide extensive user facilities

    Complete file system

    Multiple language processors

    Debugging and maintainence tools

- Allow for any number of simultaneously executing user tasks

- Avoid using processor priority to synchronize access to system data base

CHARACTERISTICS OF RSX-11M

System Features

- Supports multiprogramming

  Several tasks can be memory-resident and active at same time

  Improves efficiency of system

  Allows quick response to events

- Provides some user control of some system functions

  Scheduling
  File access

- Completely 'Event-Driven'

  No 'monolithic monitor'

  Executive entered only as a result of trap or hardware interrupt

- Disk-Based

  Tasks stored on disk until loaded
  Allows quick task startup

- Provides multi-tasking facilities

  Dynamic task requests
  Passing data between tasks
  Sharing memory areas between tasks

Major Components of RSX-11M

- Executive - Contains code to perform system functions such as task scheduling, input/output operations, etc.

- File System - Provides efficient and convenient access to data and task files.

- Utilities - Provide services needed to program and support user applications.

THE EXECUTIVE

Components

- Trap and interrupt vectors - Used by hardware to provide trap and interrupt service.

    Vectors consist of 2 words:

    Word 1 - A value to go into the PC
    Word 2 - A value to go into the PS

    When a trap or interrupt occurs:

    Current values of PC and PS are pushed onto the stack

    Values in the vector are moved into the PC and PS

- System Stack - Maintained as temporary storage for system programs.

- System Common - Contains pointers to important areas of system data base.

- Executive Code - Performs system functions.

- Dynamic Storage Area - Contains most dynamically created system data structures.

Use of Physical Memory

- Executive contained in lowest area of memory

- Size depends on features chosen at SYSGEN; not greater than 20K words

- Executive has access to I/O page in top 4K words of address space

- Some device drivers may not be included within Executive

- Some directive service routines may not be included within the Executive

16

```
                              ┌─────────────────────┐ ┐
                              │                     │ │
                              │      I/O PAGE       │ │  TOP 4K WORDS
                              │                     │ │  OF ADDRESS SPACE
                              ├─────────────────────┤ ┘
                              │                     │
                              ~        USER         ~
                                       AREA
                              ~                     ~
                              ├─────────────────────┤
                         ┐    │  DYNAMIC STORAGE    │
                         │    │      REGION         │
                         │    ├─────────────────────┤
      EXECUTIVE          │    │    EXECUTIVE        │
      (20 K WORDS        ⎬    │      CODE           │
      MAXIMUM)           │    ├─────────────────────┤
                         │    │   SYSTEM STACK      │
                         │    ├─────────────────────┤
                         ┘    │ TRAP & INTERRUPT VECTORS │
                              └─────────────────────┘
```
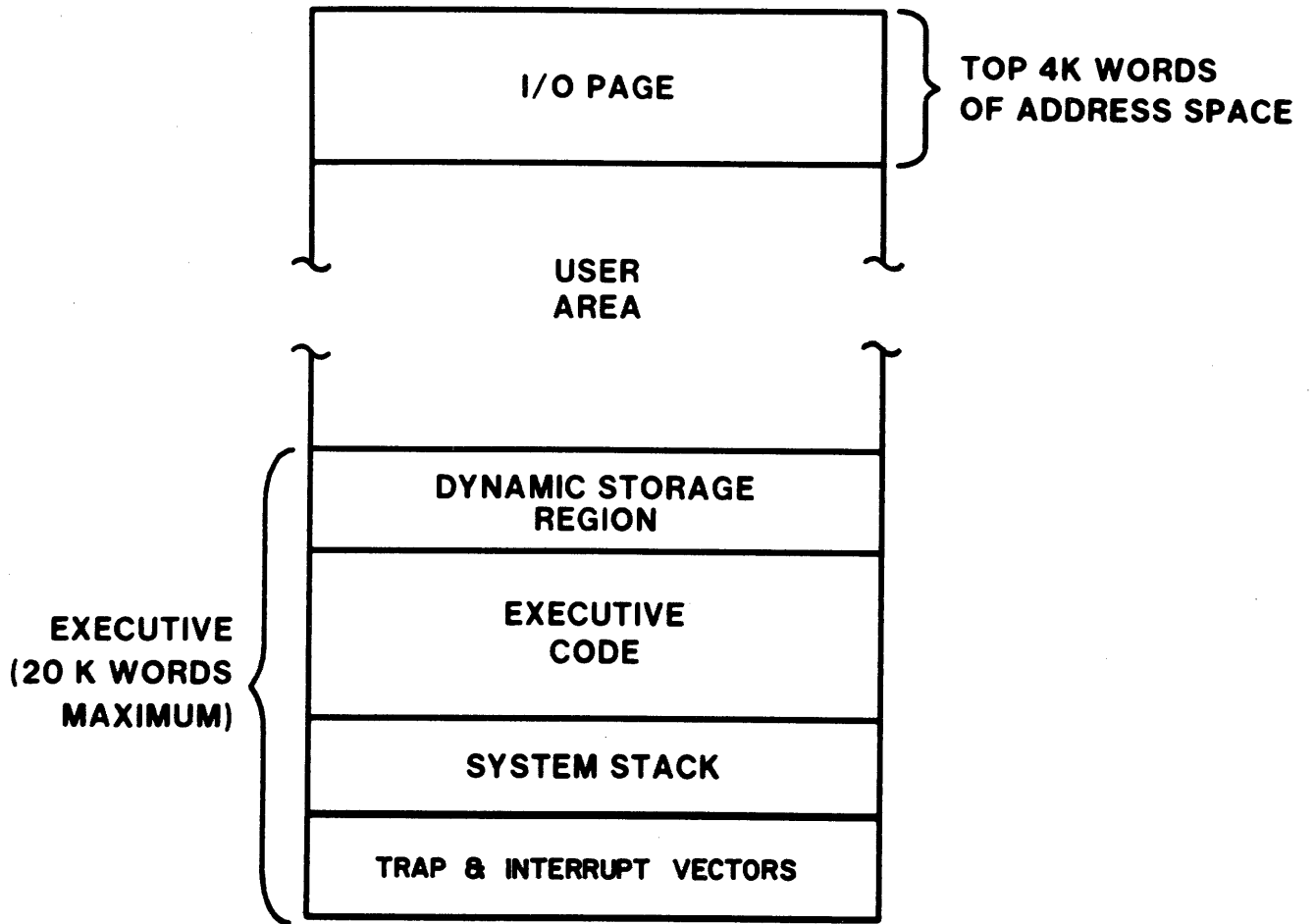
Figure 1-1   Typical Use of Physical Memory

THE FILE SYSTEM

Design Goals

- Allow programmer to organize data in a convenient manner

- Provide protection from unauthorized access

- Provide efficient use of disk space

Components

- Standard structures on disk and tape volumes

- Ancillary control processor (ACP) to provide programming interface to file structured volumes

    F11ACP - For disk volumes

    MTAACP - For magnetic tape volumes

- Application-oriented software

    FCS - File Control Services

    RMS - Record Management Services

```
┌─────────────────────┐
│       USER          │
│       TASK          │
├─────────────────────┤
│      RMS/FCS        │
└─────────────────────┘
         ▲ │
         │ │            VIRTUAL BLOCKS
         │ ▼            (VIA QIO)
┌─────────────────────┐        ┌─────────────────────┐
│                     │───────▶│       FILE          │
│     EXECUTIVE       │        │    PROCESSOR        │
│                     │◀───────│     (F11ACP)        │
└─────────────────────┘        └─────────────────────┘
         ▲ │
         │ │            LOGICAL BLOCKS
         │ ▼
┌─────────────────────┐
│      DEVICE         │
│      DRIVER         │
└─────────────────────┘
         ▲ │
         │ │            PHYSICAL BLOCKS
         │ ▼
     ╭─────────────╮
     │  PHYSICAL   │
     │   VOLUME    │
     ╰─────────────╯
```
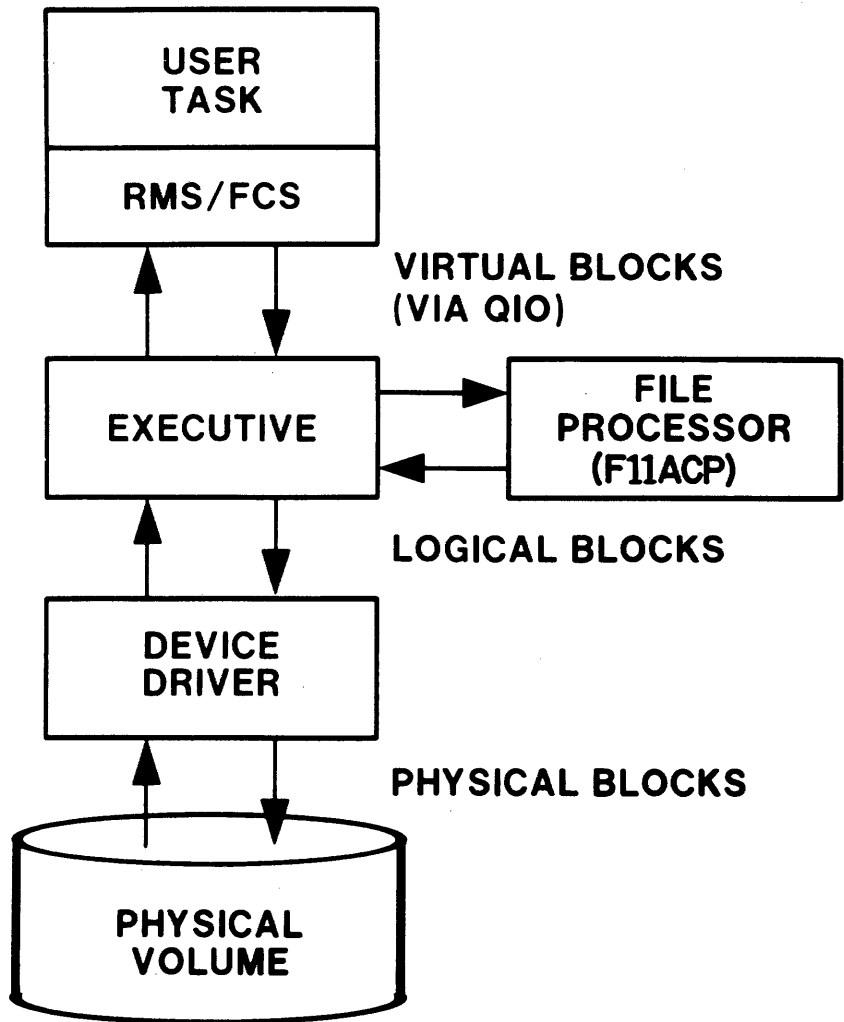
Figure 1-2   The File System

19

THE UTILITIES

- Tasks distributed with operating system

  Perform common maintenance and development functions

  Not included in memory-resident Executive

- Four general categories of utilities

  Language processors

  MACRO-11
  FORTRAN

  Program development utilities

  EDT
  EDI
  LBR

  System maintenance utilities

  BAD
  CMP
  VFY
  PAT

  File manipulation utilities

  PIP
  FLX
  DMP

THE SYSTEM DATA BASE   2

INTRODUCTION

The system data base includes the data structures which describe the current state of the system to the Executive. Every operation within the system has its effect on the system data base. Similarly, the state of the system data base at the time an operation is requested determines the success and effect of the operation.

Throughout this course, the system data base is used to explain the mechanics of various system operations. This module describes the most central system data structures, and how they are related. Future modules detail the data structures and their linkages.

OBJECTIVES

1. Describe the major system data structures associated with task management.

2. Describe the major system data structures associated with partition control.

3. Describe the major system data structures associated with I/O device control.

4. Describe how the Executive manages the Dynamic Storage Region.

STATIC AND DYNAMIC DATA STRUCTURES

● Static data structures

> Created when system is built

> Contained in Executive modules

> Example: Data structures for standard devices selected at SYSGEN are in the Executive module SYSTB.

● Dynamic data structures

> Created and eliminated as needed

> Contained in Dynamic Storage Region (DSR)

> Example: Data structures needed to define tasks to the system are in DSR.

The Dynamic Storage Region

● Contains dynamic data structures

● Space allocated and deallocated as needed

● Allocation of DSR:

> Performed by routines in module CORAL

> Length is multiple of 4 bytes

> Linked list of unallocated space is maintained by Executive

> No record of allocated space is kept

> First-fit algorithm is used

> List of preallocated I/O packets maintained if specified at SYSGEN

$CRAVL

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=0
SIZE OF FREE BLOCK (IN BYTES)

ALLOCATED
SPACE

SIZE IN BYTES INCLUDING 2 BYTES FOR POINTER AND
2 BYTES FOR SIZE FIELD

Figure 2-1   The List of Free Space in DSR

BEFORE ALLOCATION

AFTER ALLOCATION

$CRAVL

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=0
SIZE OF FREE BLOCK (IN BYTES)

$CRAVL

POINTER TO NEXT FREE BLOCK
SIZE OF FREE BLOCK (IN BYTES)

POINTER TO NEXT FREE BLOCK=0
SIZE OF FREE BLOCK (IN BYTES)

Figure 2-2   Allocating DSR

BEFORE DEALLOCATION

$CRAVL

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=0
SIZE OF FREE BLOCK (IN BYTES)

AFTER DEALLOCATION

$CRAVL

POINTER TO NEXT FREE BLOCK
SIZE OF FREE BLOCK (IN BYTES)

DEALLOCATED SPACE

POINTER TO NEXT FREE BLOCK
SIZE OF FREE BLOCK (IN BYTES)

POINTER TO NEXT FREE BLOCK=0
SIZE OF FREE BLOCK (IN BYTES)

Figure 2-3 Deallocating DSR

28

TASK-RELATED DATA STRUCTURES AND LISTS

The Task Control Block (TCB)

- Defines task to system

- Contains information controlling task execution

    Task name

    Task priority

    Local event flags

    Task status words

- Contains pointers to other data structures including:

    T.RCVL - Receive queue listhead

        Points to linked list of data blocks queued
        by the 'SEND DATA' directive.

    T.ASTL - AST queue listhead

        Points to linked list of ASTs queued to task.

    T.UCB - Address of data structure (UCB) identifying TI:

    T.TCBL - Link word for System Task Directory

    T.PCB - Address of data structure (PCB) identifying
             partition in which task executes

    T.ACTL - Link word for Active Task List

- Allows task to load and begin execution quickly

    Contains pointer to task image on disk

    Identifies partition in which to load task

UTILITY LINK WORD		T.LNK
I/O PENDING COUNT	TASK PRIORITY	T.IOC / T.PRI
POINTER TO CHECKPOINT PCB		T.CPCB
TASK NAME		T.NAM
IN RAD50		
RECEIVE QUEUE		T.RCVL
LISTHEAD		
AST QUEUE		T.ASTL
LISTHEAD		
TASK LOCAL		T.EFLG
EVENT FLAGS		
UCB ADDRESS FOR 'TI'		T.UCB
STD THREAD WORD		T.TCBL
FIRST STATUS WORD (BLOCKING BITS)		T.STAT

TCB
FIXED
AREA
PLAS AREA
OPTIONAL AREA
PARENT/OFFSPRING AREA
OTHER OPTIONS
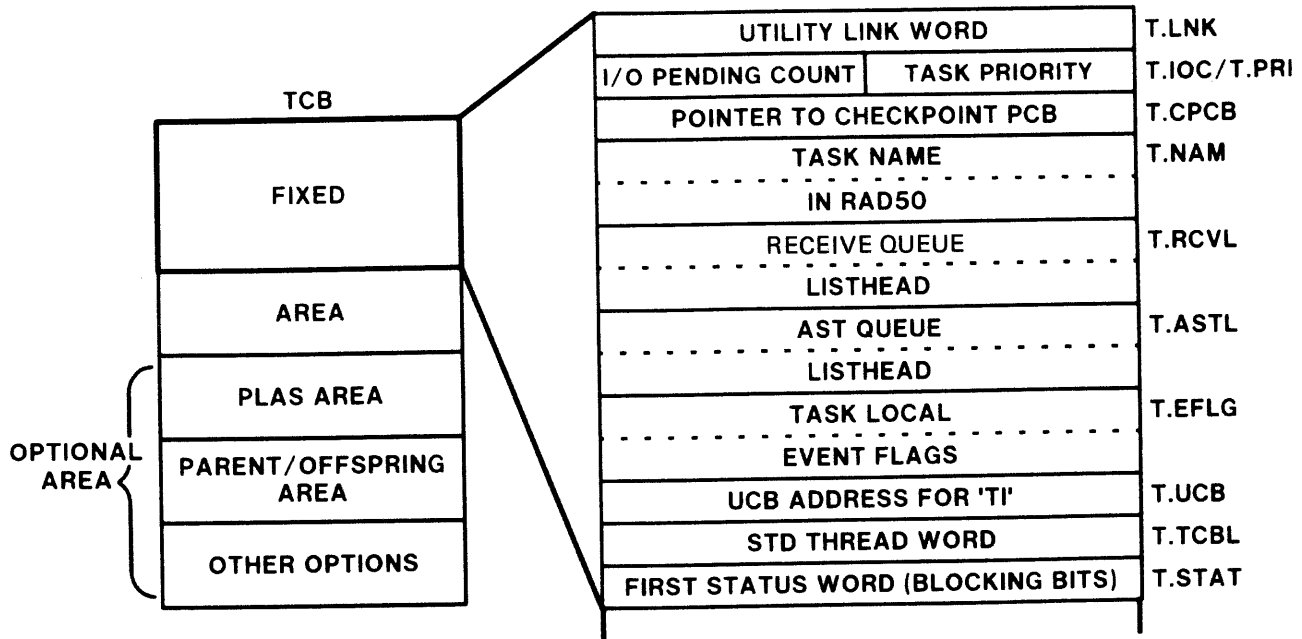
Figure 2-4   The Task Control Block
(First Part of Fixed Area)

FIRST STATUS WORD (BLOCKING BITS)		T.STAT
SECOND STATUS WORD (STATE BITS)		T.ST2
THIRD STATUS WORD (ATTRIBUTE BITS)		T.ST3
LBN	DEFAULT PRIORITY	T.LBN/T.DPRI
OF TASK LOAD IMAGE		
UCB ADDRESS OF LOAD DEVICE		T.LDV
PCB ADDRESS OF TASK PARTITION		T.PCB
MAXIMUM SIZE OF TASK IMAGE (MAPPED)		T.MXSZ
ATL THREAD WORD		T.ACTL
SPECIFIED AST LISTHEAD		T.SAST
BUFFERED I/O COUNT	UNUSED	T.TIO
TASK SIZE		T.TKSZ

TCB
FIXED
AREA
PLAS AREA
OPTIONAL AREA
PARENT/OFFSPRING AREA
OTHER OPTIONS

Figure 2-4   The Task Control Block
(Second Part of Fixed Area)
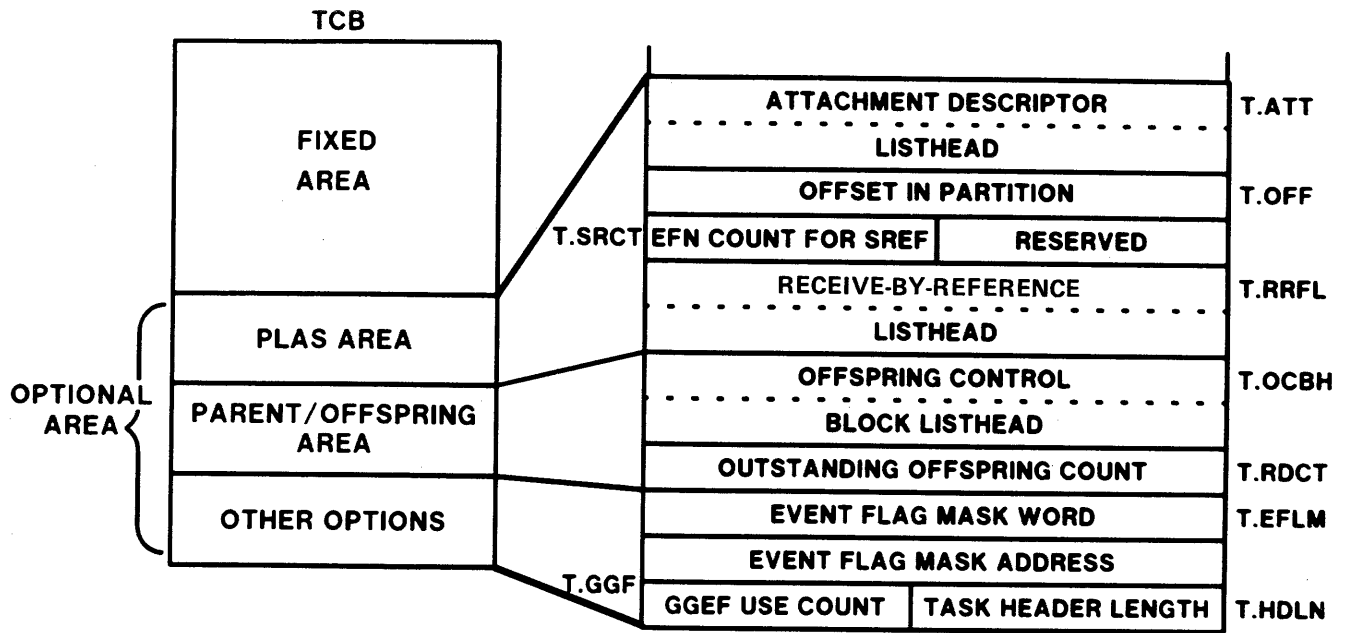
Figure 2-4   The Task Control Block
(Optional Area)

Table 2-1   Task Status Words

Offset	Bit	Symbol	Value	Meaning
T.STAT	15	TS.EXE	100000	Task not in execution
	14	TS.RDN	40000	I/O rundown in progress
	13	TS.MSG	20000	Abort message being output
	12	TS.NRP	10000	Task mapped to nonresident region
	8	TS.OUT	400	Task is out of memory
	7	TS.CKP	200	Task is being checkpointed
	6	TS.CKR	100	Task checkpointing requested

NOTE

The bits in T.STAT are called blocking bits.
They   must   all be clear for the Executive to
context switch to the task.

Offset	Bit	Symbol	Value	Meaning
T.ST2	15	T2.AST	100000	AST in progress
	14	T2.DST	40000	AST recognition disabled
	13	T2.CHK	20000	Task not checkpointable
	12	T2.CKD	10000	Checkpointing disabled
	11	T2.SEF	4000	Task stopped for event flag
	10	T2.FXD	2000	Task fixed in memory
	9	T2.TIO	1000	Task engaged in terminal I/O
	8	T2.CAF	400	Dynamic checkpoint space allocation failure
	7	T2.HLT	200	Task is being halted
	6	T2.ABO	100	Task marked for abort
	5	T2.	40	Saved T2.STP on AST in progress
	4	T2.STP	20	Task stopped
	3	T2.	10	Saved T2.SPN on AST in progress
	2	T2.SPN	4	Task suspended
	1	T2.	2	Saved T2.WFR on AST in progress
	0	T2.WFR	1	Task in wait for state

Offset	Bit	Symbol	Value	Meaning
T.ST3	15	T3.ACP	100000	Ancillary control processor
	14	T3.PMD	40000	Inhibit post mortem dump
	13	T3.REM	20000	Remove task on exit
	12	T3.PRV	10000	Task is privileged
	11	T3.MCR	4000	Task requested as external MCR function
	10	T3.SLV	2000	Task is slaved
	9	T3.CLI	1000	Task is a command line interpreter
	7	T3.NSD	200	Task does not allow send data
	6	T3.CAL	100	Checkpoint space allocated in task image
	5	T3.ROV	40	Task has resident overlays

The System Task Directory (STD)

- Linked list of TCBs

- Ordered by descending default priority

- Begins at listhead $TSKHD in System Common

- Linked through location T.TCBL in the TCB

- 'Null Task' TCB is last entry

   'Null Task' used to anchor system task lists

   When 'Null Task' is scheduled, the idle loop is performed
   until another task becomes ready to run

   'Null Task' TCB in System Common

   'Null Task' never executes

- TCB is inserted when task is installed

- New TCB inserted after others of same priority

- System maintains pointers (in SYSCM) to special tasks in STD

   $TKTCB - Current task TCB

   $TKNPT - Pointer to TCB of task termination task TKTN

   $MCRPT - Pointer to TCB of MCR task

   $LDRPT - Pointer to loader TCB

$TSKHD

T.PRI 248
T.TCBL
T.DPRI 248
T.ACTL

TCB OF
(TYPICALLY)
LOADER TASK

$RQSCH

T.PRI 160
T.TCBL
T.DPRI 160
T.ACTL

T.PRI 100
T.TCBL
T.DPRI 100

T.PRI 50
T.TCBL
T.DPRI 50
T.ACTL

$TKTCB

T.PRI 130
T.TCBL
T.DPRI 50
T.ACTL

T.PRI 0
T.TCBL-0
T.ACTL

THIS TCB IS LOCATED IN THE
EXECUTIVE MODULE SYSCM.
IT IS THE NULL TASK'S TCB.
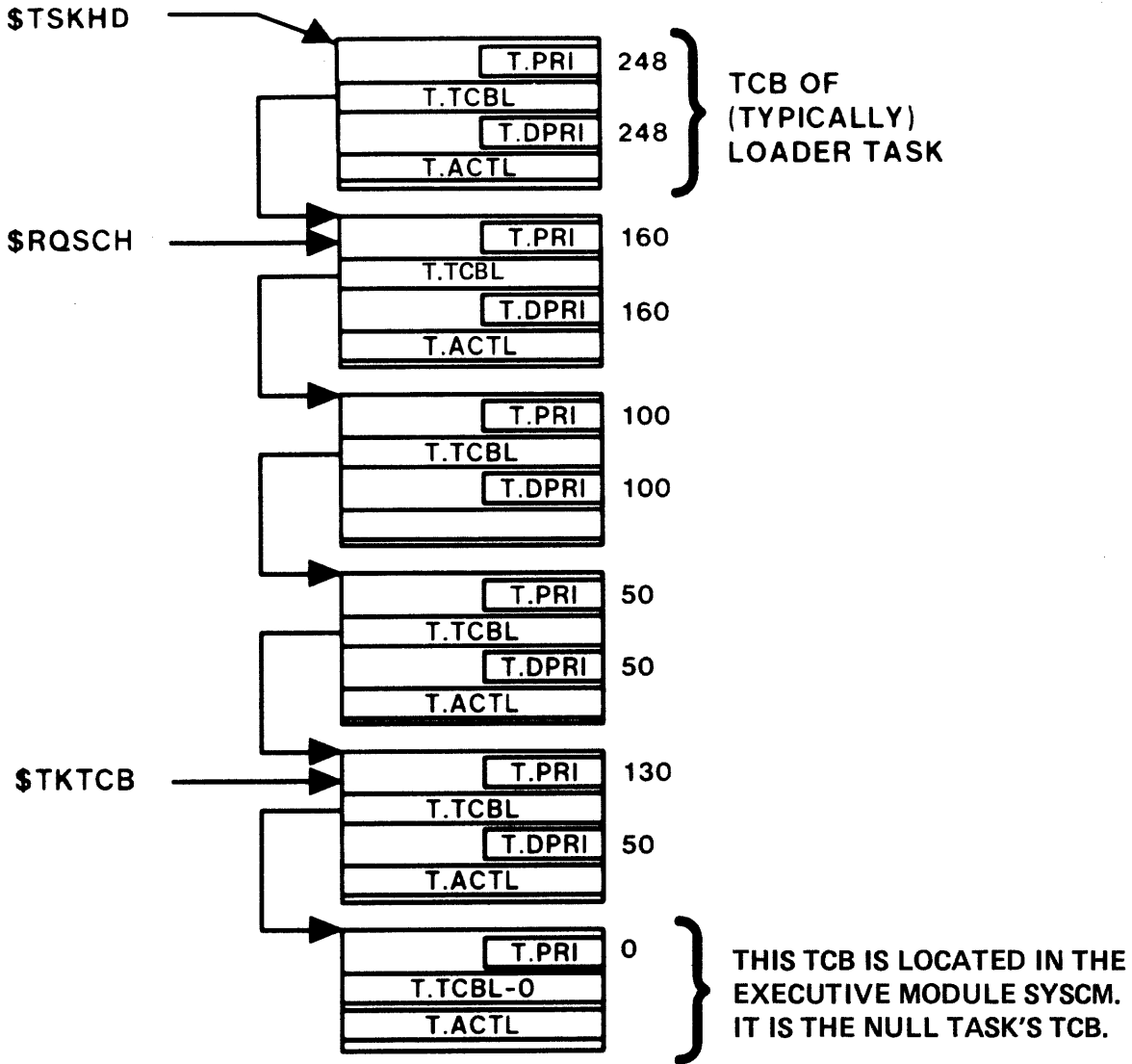
Figure 2-5   The System Task Directory

The Active Task List (ATL)

- Linked list of TCBs

- Ordered by descending priority

- Begins at $ACTHD in System Common

- Linked through T.ACTL in TCB

- 'Null Task' TCB is last entry

- TCB inserted when requested; Task becomes active

- New TCB inserted after others of same priority
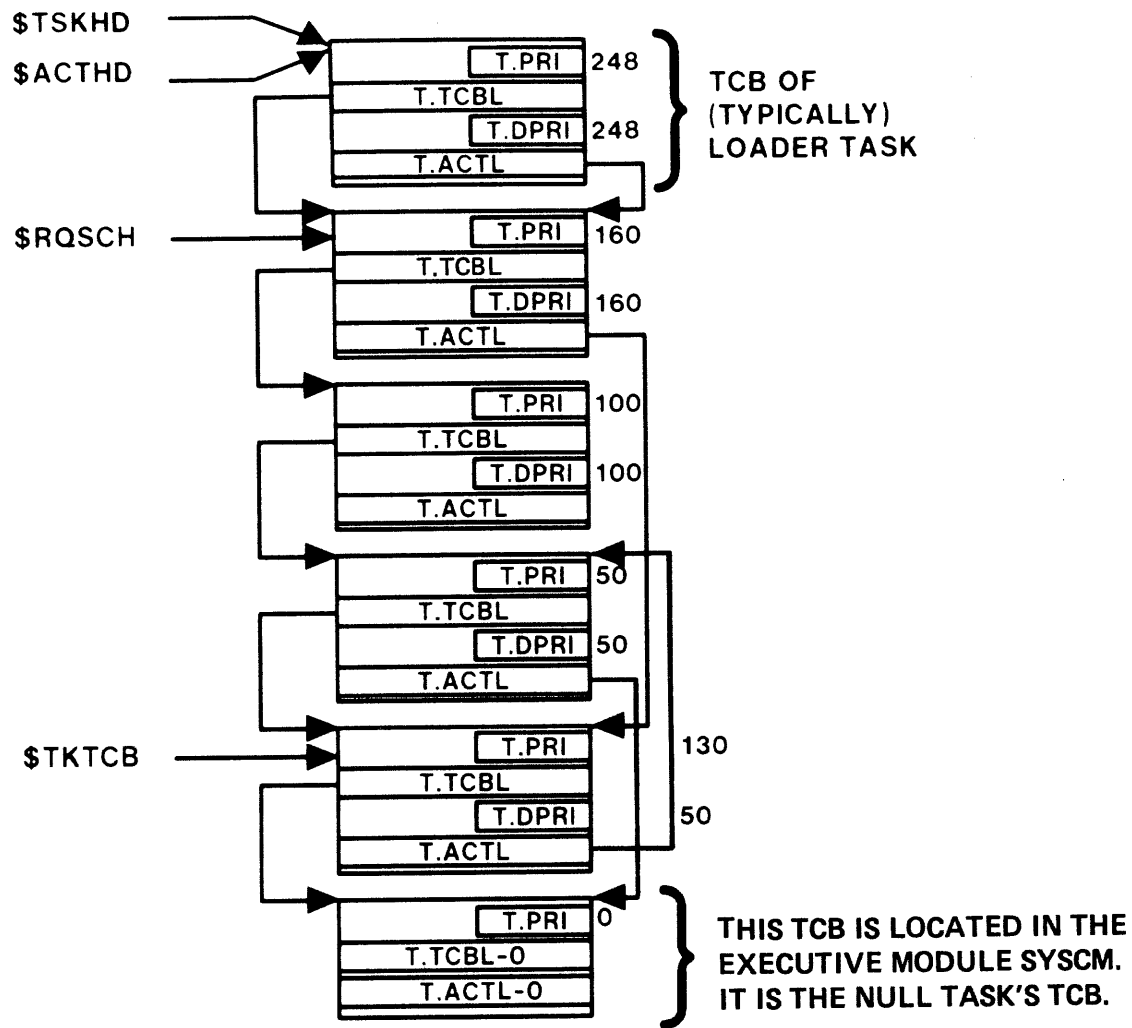
Figure 2-6   The Active Task List

PARTITION-RELATED DATA STRUCTURES AND LISTS

The Partition Control Block (PCB)

- Defines partition to system

- Contains information describing partition

    Partition name

    Physical starting address (32 word blocks number)

    Size of partition (in 32 word blocks)

    Partition busy word

    Partition status word

    Protection word

    Current UIC of owner task

- Pointers to other data structures

    If subpartition PCB, pointer to the main partition PCB

    Pointer to next subpartition PCB

    Pointer to queue of TCBs of tasks waiting for space in partition

    Pointer to attachment descriptor block for task

    Pointer to task header of resident task

LINK TO NEXT PARTITION PCB		0	P.LNK
P.IOC I/O COUNT	PRIORITY OF PARTITION	2	P.PRI
PARTITION NAME		4	P.NAM
(IN RADIX-50)		6	
POINTER TO NEXT SUB-PARTITION		10	P.SUB
POINTER TO MAIN PCB		12	P.MAIN
PHYSICAL START ADDRESS OF PARTITION		14	P.REL[1],P.HDR[2]
SIZE OF PARTITION		16	P.BLKS[1],P.SIZE[1]
PARTITION WAIT QUEUE LIST HEAD		20	P.WAIT
PARTITION SWAP SIZE (SYSTEM ONLY)		22	P.SWS2[3]
PARTITION BUSY FLAGS		24	P.BUSY (TWO BYTES)
TCB ADDR. OF OWNER/CURRENT UIC OF OWNER TASK		26	P.TCB/P.OWN
PARTITION STATUS FLAGS		30	P.STAT
POINTER TO TASK HEADER		32	P.HDR[2]
PROTECTION WORD FOR PARTITION		34	P.PRO[4]
ATTACHMENT DESCRIPTOR		36	P.ATT[5]
LIST HEAD		40	

1. 32 WORD BLOCKS FOR MAPPED SYSTEMS
   BYTES FOR UNMAPPED SYSTEMS
2. P.HDR=P.REL IN UNMAPPED SYSTEMS
3. FOR SUB-PARTITIONS OF SYSTEM CONTROLLED PARTITIONS
4. MAPPED SYSTEMS ONLY
5. MAPPED SYSTEM WITH PLAS ONLY

Figure 2-7 The Partition Control Block

Table 2-2   Partition Status Word

Bit	Symbol	Value	Meaning
15	PS.OUT	100000	Partition is out of memory
14	PS.CKP	40000	Partition checkpoint in progress
13	PS.CKR	20000	Partition checkpoint requested
12	PS.CHK	10000	Partition is not checkpointable
11	PS.FXD	4000	Partition is fixed
10	PS.PER	2000	Parity error in partition
9	PS.LIO	1000	Marked by shuffler for long I/O
8	PS.NSF	400	Partition is not shufflable
7	PS.COM	200	Library or common block
6	PS.PIC	100	Position independent library of common
5	PS.SYS	40	System controlled partition
4	PS.DRV	20	Driver is loaded in partition
3	PS.DEL	10	Partition should be deleted when not attached
2	PS.APR	4	Starting APR
1		2	Starting APR
0		1	Starting APR

The Partition List

- Linked list of PCBs

- Linked in order of increasing base address

- Begins at location $PARHD in System Common

- Linked through P.LNK in PCB

- PCB created and inserted in list by 'SET /MAIN' or 'SET /SUB' MCR or VMR command

- PCBs of subpartitions

    Task partitions

    Linked into list using P.LNK
    Linked to PCB of main partition using P.SUB

    System-controlled partitions

    Linked to main partition PCB using P.SUB
    Not linked through P.LNK

PCBS OF DYNAMICALLY ALLOCATED
SUBPARTITIONS ARE CHAINED ACCORDING
TO THE BASE ADDRESSES OF THEIR SUBPARTITIONS
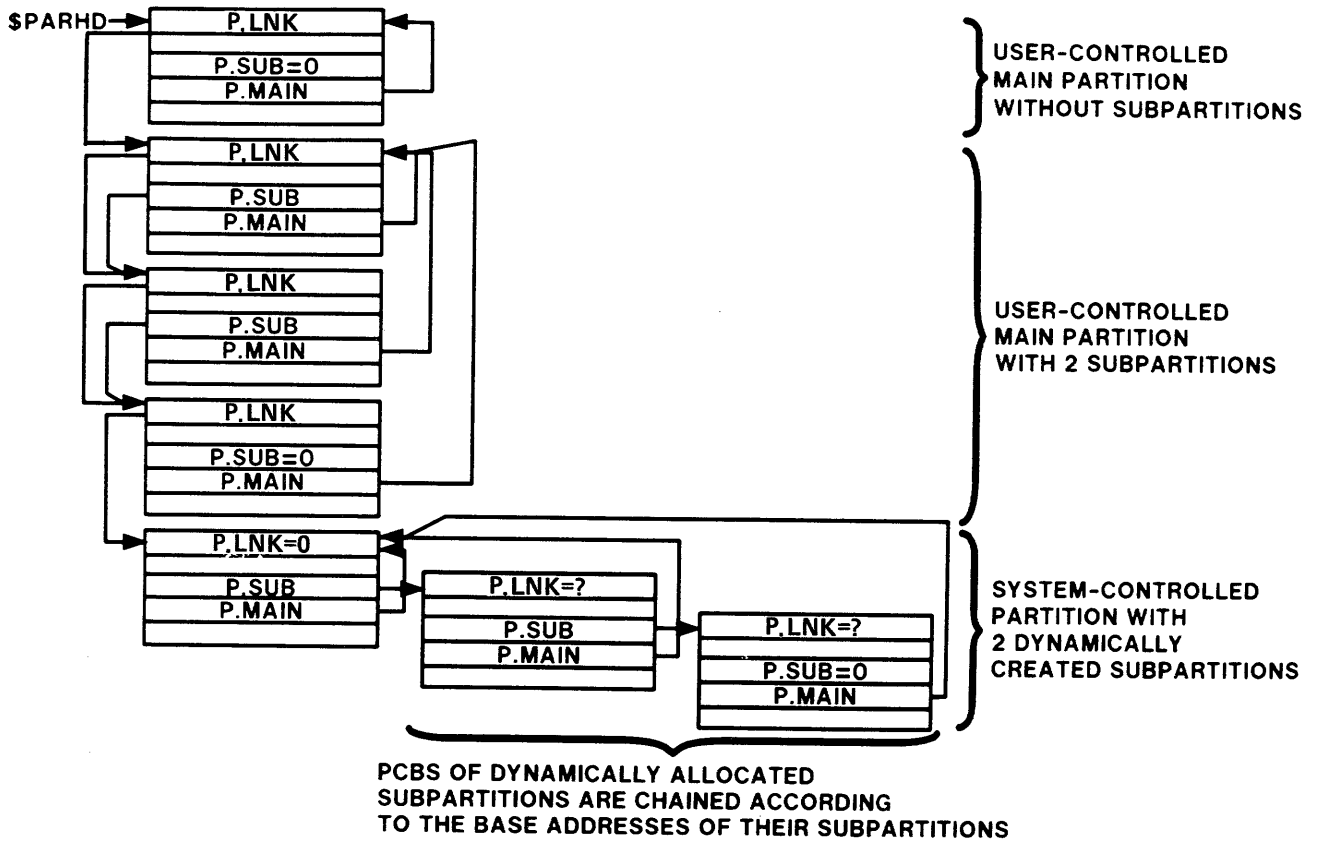
Figure 2-8   The Partition List
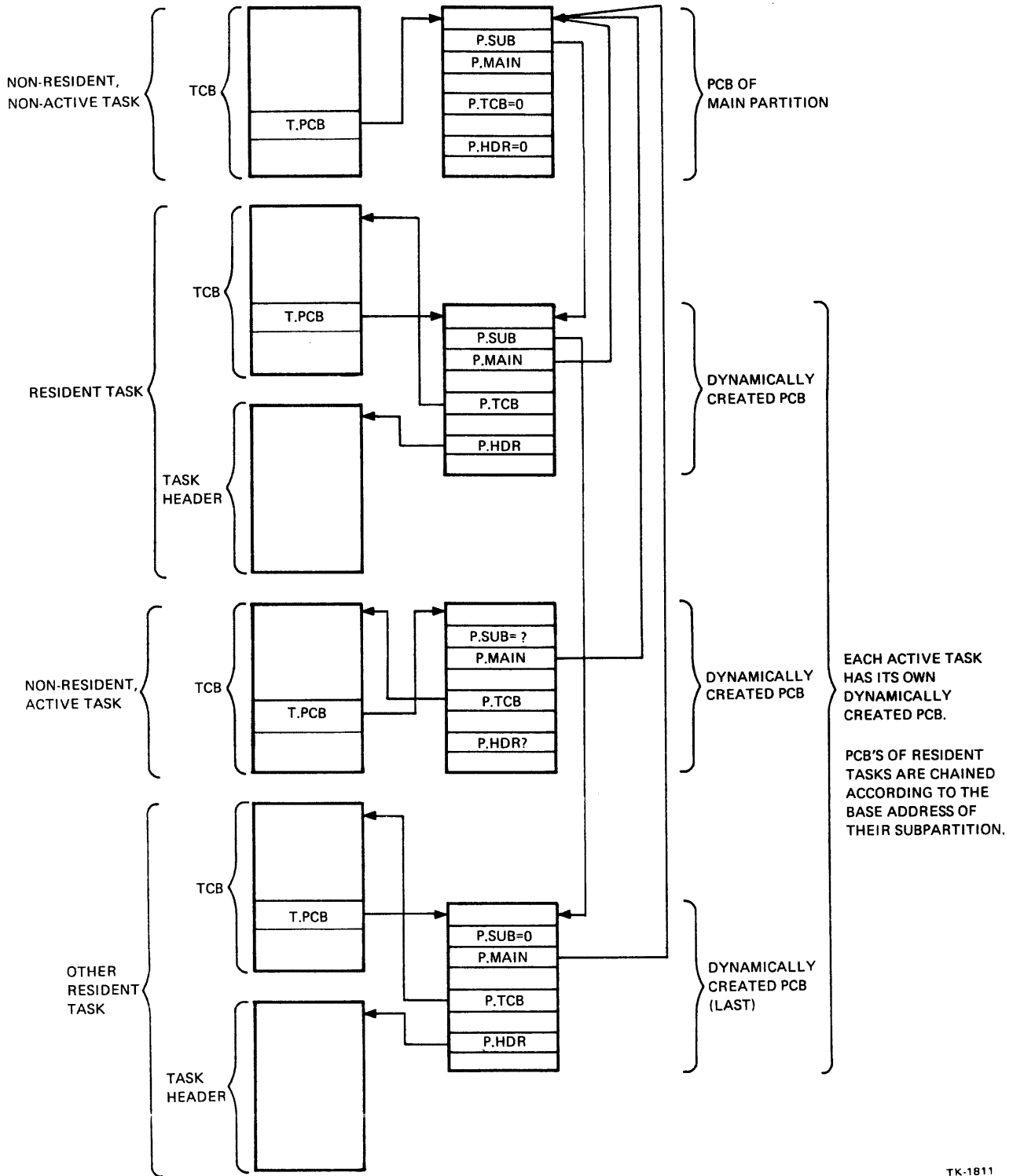
Figure 2-9   Relationship Between Task and Partition Structures

DEVICE-RELATED DATA STRUCTURES AND LISTS

The Device Control Block (DCB)

- Describes physical device type

- Contains information describing device

    Generic device name in ASCII (D.NAM)

    Physical units in system of this type (D.UNIT)

    Device functions supported (D.MSK)

- Pointer to data structures describing physical units (D.UCB)

- Pointer to entry points in device driver (D.DSP)

LINK TO NEXT DCB	0	D.LNK
POINTER TO FIRST UCB OF THIS DEVICE TYPE	2	D.UCB
GENERIC DEVICE NAME (IN ASCII)	4	D.NAM
HIGHEST UNIT NUMBER LOWEST UNIT NUMBER	6	D.UNIT
LENGTH OF EACH UCB OF THIS DEVICE TYPE	10	D.UCBL
POINTER TO DRIVER DISPATCH TABLE	12	D.DSP
LEGAL FUNCTION MASK CODES 0-15	14	D.MSK
CONTROL FUNCTION MASK CODES 0-15	16	
NO-OP FUNCTION MASK CODES 0-15	20	
ACP FUNCTION MASK CODES 0-15	22	
LEGAL FUNCTION MASK CODES 16-31	24	
CONTROL FUNCTION MASK CODES 16-31	26	
NO-OP FUNCTION MASK CODES 16-31	30	
ACP FUNCTION MASK CODES 16-31	32	
PCB ADDRESS OF LOADABLE DRIVER	34	D.PCB

Figure 2-10   The Device Control Block

43

The Device List

- Linked list of DCBs

- Ordered by inclusion in SYSTB and by order of loading

- Begins at listhead $DEVHD in SYSCM

- Linked through location D.LNK in DCB

```
SYSCM  ──────▶  ┌──────────────┐
                │    D.LNK     │
 ($DEVHD)       ├──────────────┤
                │ DCB FOR LP:  │
                │              │
                │              │
                └──────────────┘

                ┌──────────────┐
                │    D.LNK     │
                ├──────────────┤
                │ DCB FOR DB:  │
                │              │
                │              │
                └──────────────┘

                ┌──────────────┐
                │    D.LNK     │
                ├──────────────┤
                │ DCB FOR DK:  │
                │              │
                │              │
                └──────────────┘
```
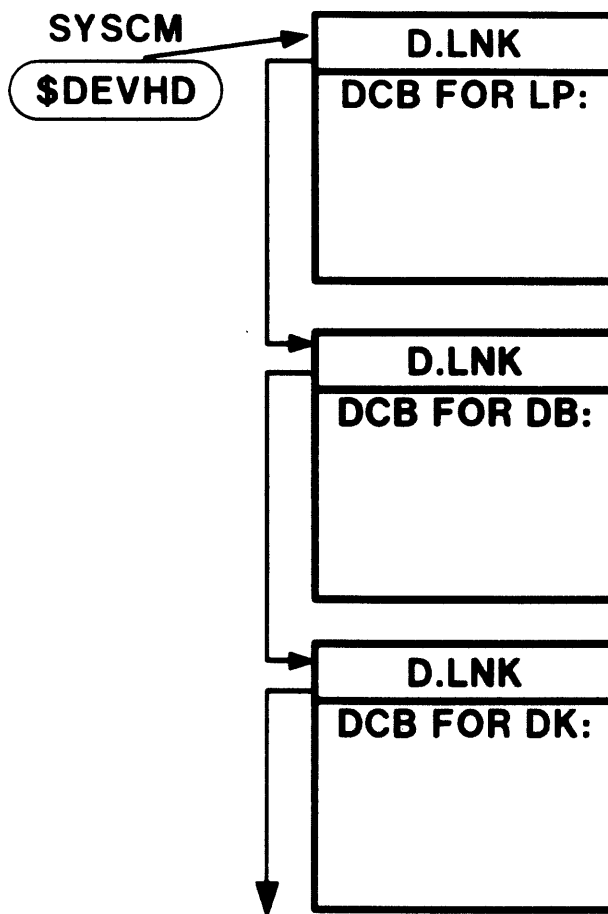
Figure 2-11   The Device List

44

Other Structures

● The Unit Control Block (UCB)

    One UCB per physical unit of type

    Size depends on device type

    Contains data used to describe physical unit

● The Status Control Block (SCB)

    One SCB per controller or per line if multiplexer

    Size depends on device type

    Contains data needed to control I/O operations
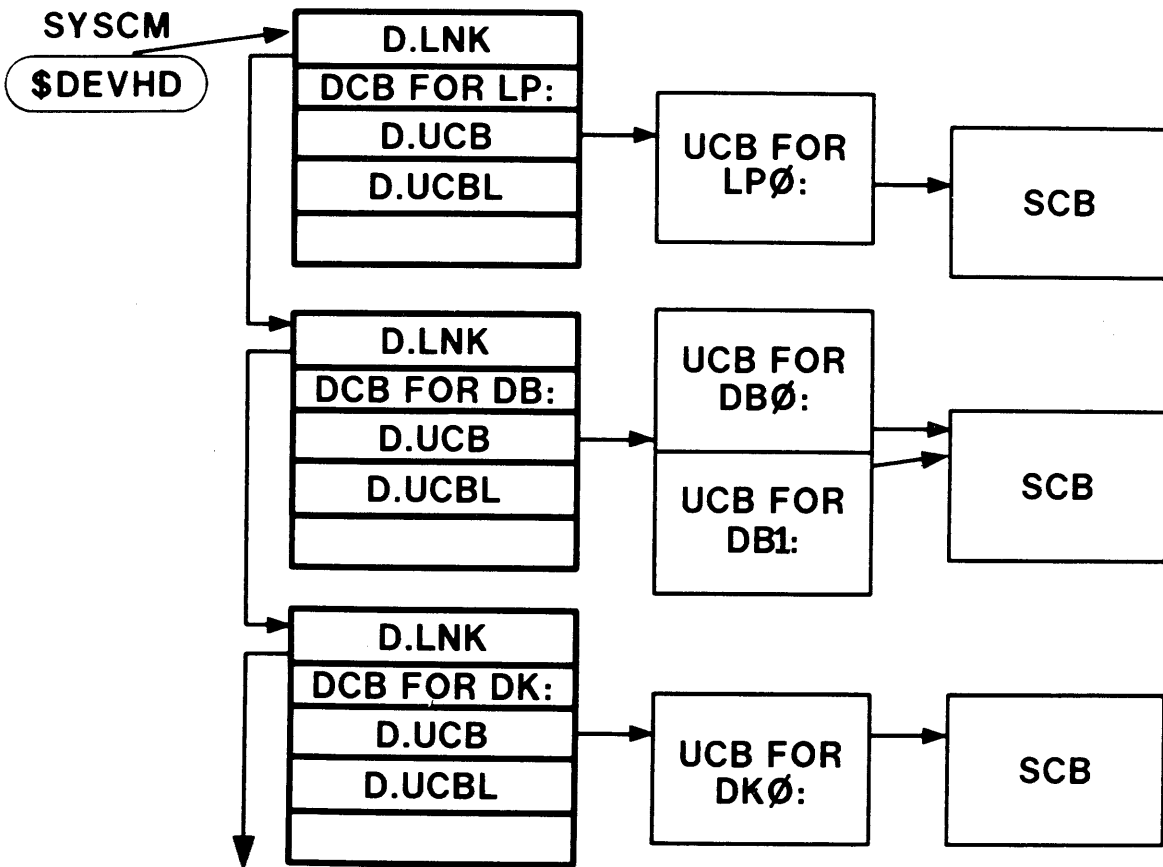
Figure 2-12 Other Data Structures

THE CLOCK QUEUE

The Clock Queue Control Block

●   Describes time-dependent request

●   Contains eight words;   last three depends on type of request

●   Five types of requests

> Mark Time

> Task request with periodic rescheduling

> Single shot task request

> Single shot internal subroutine (two types)

POINTER TO NEXT CLOCK QUEUE CONTROL BLOCK		0 C.LNK
EVENT FLAG NUMBER (MARK TIME ONLY)	REQUEST TYPE	2 C.RQT
TCB ADDRESS OR SYSTEM SUBROUTINE ADDRESS		4 C.TCB
ABSOLUTE TIME ——— WHEN ———	LOW ORDER PART	6 C.TIM
REQUEST COMES DUE	HIGH ORDER PART	10
———TYPE		12
DEPENDENT		14
PARAMETERS———		16

C.EFN 3
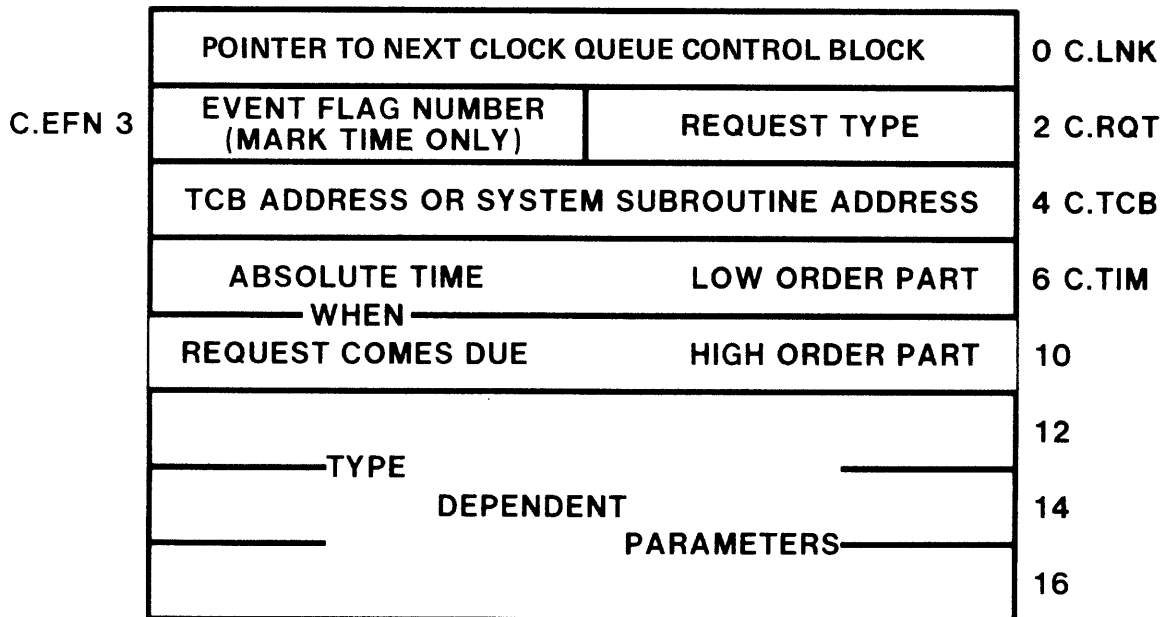
Figure 2-13   The Clock Queue Control Block

46

The Clock Queue

● Linked list of Clock Queue Control Blocks

● List ordered by increasing absolute time

● Scanned by code in TDSCH when clock interrupt occurs

SYSCM   **DSR**

$CLKHD ──────────────► ┌──────────────┐
                       │   C.LNK   │
                       ├──────────────┤
                       │   │
                       │   │
                       │   │
                       │   │
                       └──────────────┘

                       ┌──────────────┐
                       │   C.LNK   │
                       ├──────────────┤
                       │   │
                       │   │
                       │   │
                       │   │
                       └──────────────┘

                       ┌──────────────┐
                       │   C.LNK=0   │
                       ├──────────────┤
                       │   │
                       │   │
                       │   │
                       │   │
                       └──────────────┘
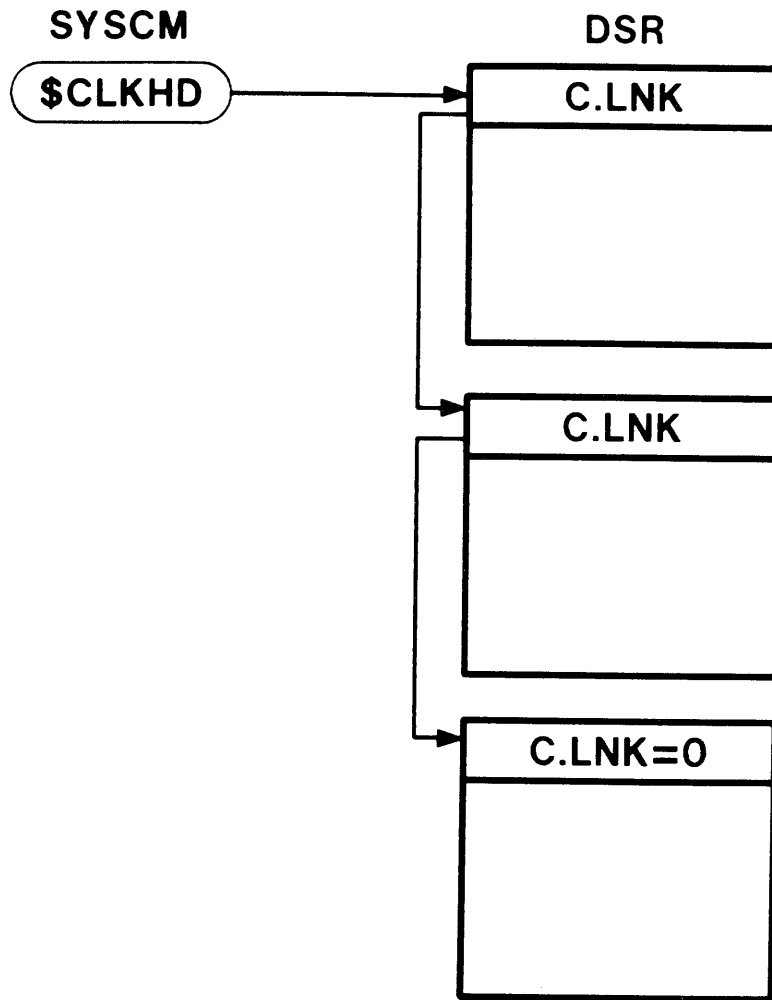
Figure 2-14   The Clock Queue

EXECUTIVE MODULES

3

INTRODUCTION

The Executive is composed of a number of modules. Each module contains code which performs a specific function or class of related functions. To effectively function as a systems programmer, you must be able to locate this code. This module provides a quick overview of the major modules in the Executive.

OBJECTIVES

1. Recognize the general function of a module (i.e., general, directive processing, or device driver).

2. Locate the code which performs a given Executive function.

OVERVIEW

- Over 90 possible modules in the Executive

- Modules conditionally assembled using symbols contained in [11,10]RSXMC.MAC

- Three types of modules:

General Executive modules

   Have names which relate to function

   Example: SYSCM - System Common

Directive Processing modules

   Have 5 character names

      First two characters are 'DR'

      Last three characters indicate function

   Example: DRABO - Module for processing the abort directive

Device Driver modules

   Have 5 character names

      First two characters indicate device type

      Last three characters are 'DRV'

   Example: DKDRV - Device driver for RK05 disk drives

PRINCIPAL EXECUTIVE MODULES

System Common (SYSCM)

- Contains pointers to system lists

      $CRAVL - Pointer to first block of free pool space
      $TSKHD - Pointer to STD
      $ACTHD - Pointer to ATL
      $PARHD - Pointer to Partition List
      $DEVHD - Pointer to Device List
      $GGEF - Pointer to Group Global Event Flag List
      $CLKHD - Pointer to Clock Queue
      $FRKHD - Pointer to Fork Queue
      $CFLPT - Pointer to Checkpoint File PCB List
      $UMRWT - Pointer to UMR wait queue

- Contains pointers to specific data blocks

      $HEADR - Pointer to header of current task
      $TKNPT - Pointer to TCB of TKTN
      $SHFPT - Pointer to TCB of shuffler
      $TKTCB - Pointer to TCB of current task
      $RQSCH - Pointer to TCB of requested task
      $MCRPT - Pointer to TCB of MCR
      $LDRPT - Pointer to TCB of loader

- General system data

      $CMBEG - The TCB of the null task
      $UMRHD - First UMR mapping assignment block
      $CURPR - Current task priority
      $COMEF - Common event flags
      $PWRFL - Powerfail recovery request flag
      $ABTIM - Absolute time counter
      $STKDP - Stack depth indicator
      $INTCT - Clock interrupt ticks count
      $DYPMN - Days per month table
      $BTMSK - Bit mask table
      $IOABM - I/O active bitmap
      $TKPS - Ticks per second
      $CXDBL - Context switching disabled flag

System Entrance and Exit Routines (SYSXT)

- Contains routines used for entering and exiting the Executive

    $DIRSV – Called when Executive entered by system trap
    $FORK – Used to create FORK process by driver
    $FORK1 – Used to create FORK process by non-driver
    $FORK2 – Used to create FORK process by CINT$ routine
    $INTSV – Called when Exec entered by interrupt
    $INTSE – Called when Exec entered by interrupt (Error
             Logging)
    $INTSC – Called when Exec entered by interrupt (CINT$)
    $INTXT – Called to exit the Exec after an interrupt
    $DIRXT – Called to exit the Exec after a system trap
    $SWSTK – Called to enter the Exec from a privileged task

- Contains code for the system 'idle loop'

DSR Allocation Routines (CORAL)

- Contains routines used to allocate space in DSR

    $ALOCB – Allocates a block of specified size from DSR
    $ALCLK – Allocates a clock queue control block from DSR
    $DECLK – Deallocates a clock queue control block
    $ALPKT – Allocates a 'send' or I/O request block
    $DEPKT – Deallocates a 'send' or I/O request block
    $DEACB – Deallocates a block

I/O Related Routines (IOSUB)

● Contains routines used in I/O-related processing

        $GTPKT - Gets I/O packet from I/O queue
        $GSPKT - Gets I/O packet
        $IODON - Handles common I/O completion

PLAS Routines (PLSUB)

● Contains subroutines used to manipulate Program Logical Address Space (PLAS)

        $SRNAM - Searches for specified partition
        $CKACC - Checks if desired access to region is allowed
        $CRATT - Creates Attachment Descriptor Block (ADB)
        $SRATT - Searches for specified ADB
        $SRWND - Searches for specified address window
        $UNMAP - Unmaps address window

Queue Manipulation Routines (QUEUE)

● Contains subroutines used to manipulate queues

        $CLINS - Inserts an entry into the clock queue
        $CLRMV - Removes an entry from the clock queue
        $QINSF - Inserts into FIFO queue
        $QINSP - Inserts into priority queue
        $QMCRL - Queues MCR command line
        $QRMVF - Removes from front of queue
        $QRMVT - Removes next entry with specified TCB address

Task Request Related Routines (REQSB)

- Contains subroutines used to handle task requests

```
$ABCTK - Aborts current task
$ABTSK - Aborts specified task
$BILDS - Sets up task stack and initialize task header
$ACTTK - Puts task in Active Task List (ATL)
$SETCR - Sets conditional scheduling request
$SETF  - Sets event flag
$DASTT - Declares a non-I/O related AST
$QASTT - Queues an AST to a specified task
$SRAST - Searches for specified type of AST
$SRSTD - Searches STD for specified task
$ACTRM - Removes task from ATL
$STPCT - Stops current task
$STPTK - Stops specified task
$RLPAR - Releases task partition
$RLPR1 - Releases specified partition
$NXTSK - Assigns next task to partition
$FNDSP - Finds space in system controlled partitions
$TSTCP - Tests if checkpoint should be initiated
$ICHKP - Initiates checkpoint of specified task
$CHKPT - Checkpoint task
$LOADT - Puts task in loader queue
$TSKRT - Requests task execution
$MAPTK - Maps task address window
```

SST Service Routines (SSTSR)

- Contains routines to service Synchronous System Traps (SST)

```
$EMSST - Non-RSX EMT instruction
$ILINS - Illegal or reserved instruction
$IOTRP - IOT instruction
$SGFLT - Memory protect violation
$TRACE - T-bit trap or BPT instruction
$TRP04 - Odd address, Non-existent memory, etc.
$SSTXT - Common SST exit routine
```

Time Dependent Scheduling (TDSCH)

● Contains routines to handle time dependent scheduling

● Entered from clock interrupt

Updates system time and date
Manages clock queue
Checks for device timeouts
Manages Round Robin scheduling
Manages Executive level disk swapping

System Initialization (INITL)

● Executes only for system which has not been saved

Contains the Executive transfer address location (\$INITL)

● Performs system initialization functions

Writes load device and image file information into SYSCM
Initializes Processor Status Word
Initializes Memory Management registers (Mapped system)
Turns on memory management hardware (Mapped system)
Initializes Unibus Mapping Registers (11/44, 11/70)
Calculates size of physical memory
Verifies the existence of the external directive commons
Verifies the presence of the floating point processor
Verifies the presence of EIS
Verifies that all selected devices are online
Verifies the presence of the watchdog timer
Sets up system clock

● Deallocates its space to DSR

● Values initialized by INITL are written to system image by SAV task

Table 3-1   Executive Modules by Function

Function	Modules
Scheduling	
Task Scheduling	TDSCH - Time-Dependent Scheduling
	SYSXT - Normal Scheduling
Round Robin Scheduling	TDSCH
Disk-Swapping	TDSCH
Idle Loop	SYSXT
Interrupt Processing	
Clock Interrupts	TDSCH
I/O Interrupts	XXDRV - 'XX' Indicates Device
System Entrance	SYSXT
System Exit	SYSXT
Trap Processing	
Synchronous System Traps	SSTSR
'EMT' Traps	DRSUB
	DRDSP
'TRAP' Traps	DRSUB
	DRDSP
System Entrance	SYSXT
System Exit	SYSXT
Context Switching	SYSXT
Data Movement	BFCTL

Table 3-1 Executive Modules by Function (Cont)

Function	Modules
DSR Allocation	
For Clock Queue	CORAL
For FIFO Queues	CORAL
For Priority Queues	CORAL
For CLI Messages	CORAL
For Error Messages	ERROR
Error Logging	ERROR
I/O Processing	
I/O Parameter Block	IOSUB
UMR Allocation	IOSUB
I/O Completion	IOSUB

DIRECTIVE MODULES

Resident Directive Module (DRSUB)

- Contains entry points for routines entered by system traps

      $TRTRP - Handles TRAP instruction
      $EMTRP - Handles EMT instruction

- Contains commonly used routines

      $ELGEF - Eliminates Group Global Event Flag block
      $DEAGF - Deaccesses group global event flags
      $DRDSE - Declares significant event
      $DRWSE - Waits for significant event
      $MPXC1 - Maps the first directive common
      $MPXC2 - Maps the second directive common
      $DRQRQ - Queues an I/O request and calls driver
      $GTUCB - Gets UCB address for given device and unit
      $OTHR1 - Calls directive routine in second directive
               common

Directive Dispatcher (DRDSP)

- Contains code to process EMT traps

- Determines if EMT is

      Directive call

      Switch to Kernel mode

      Non-RSX EMT instruction

- Contains table defining directives in system

Table 3-2 Directive Processing Modules

Entry	Module	Module	Function
ABRT$	$DRABO	DRABO	Abort Task
ALTP$	$DRAP1	DRATP	Alter Priority
ALUN$	$DRASG	DRASG	Assign LUN
ASTX$S	$DRAXT	DRATX	AST Service Exit
ATRG$	$DRATR	DRREG	Attach Region
CINT$	$DRCIN	DRCIN	Connect to Interrupt Vector
CLEF$	$DRCEF	DRSED	Clear Event Flag
CMKT$	$DRCMS	DRCMT	Cancel Mark Time Requests
CNCT$	$DRCNC	DRSPW	Connect
CRAW$	$DRCRW	DRMAP	Create Address Window
CRGF$	$DRCRE	DRGEF	Create Group Global Event Flags
CRRG$	$DRCRR	DRREG	Create Region
CSRQ$	$DRCSR	DRCMT	Cancel Time Based Requests
DECL$S	$DRDSE	DRSUB	Declare Significant Event
DSAR$S	$DRDAR	DRDAR	Disable AST Recognition
DSCP$S	$DRDCP	DRDCP	Disable Checkpointing
DTRG$	$DRDTR	DRREG	Detach Region
ELAW$	$DRELW	DRMAP	Eliminate Address Window
ELGF$	$DRELE	DRGEF	Eliminate Group Global Event Flags
EMST$	$DREMS	DRSPW	Emit Status
ENAR$S	$DREAR	DRDAR	Enable AST Recognition
ENCP$S	$DRECP	DRDCP	Enable Checkpointing
EXIF$	$DREIF	DREIF	Exit If
EXIT$S	$DREXT	DREIF	Task Exit
EXST$	$DREXS	DRSPW	Exit With Status
EXTK$	$DREXP	DREXP	Extend Task
GCCI$	$DRGCL	DRGCL	Get command for command interpreter
GCII$	$DRCLI	DRCLI	Get command interpreter information
GLUN$	$DRGLI	DRGLI	Get LUN Information
GMCR$	$DRGCL	DRGCL	Get MCR Command Line
GMCX$	$DRGMX	DRMAP	Get Mapping Context
GPRT$	$DRGPP	DRGPP	Get Partition Parameters
GREG$	$DRGPP	DRGPP	Get Region Parameters
GSSW$S	$DRGSS	DRGSS	Get Sense Switches
GTIM$	$DRGTP	DRGTP	Get Time Parameters
GTSK$	$DRGTK	DRGTK	Get Task Parameters
MAP$	$DRMAP	DRMAP	Map Address Window
MRKT$	$DRMKT	DRMKT	Mark Time
QIO$	$DRQIO	DRQIO	Queue I/O Request
QIOW$	$DRQIO	DRQIO	Queue I/O Request and Wait
RCST$	$DRRCS	DRRAS	Receive Data or Stop
RCVD$	$DRREC	DRRAS	Receive Data
RCVX$	$DRREC	DRRAS	Receive Data or Exit
RDAF$	$DRRAF	DRRES	Read All Event Flags
RDXF$	$DRRAF	DRRES	Read Extended Event Flags

Table 3-2   Directive Processing Modules (Cont)

	Entry	Module	Function
RPOI$	$DRREQ	DRSPW	Request and pass offspring information
RQST$	$DRREQ	DRSPW	Request Task
RREF$	$DRRRF	DRMAP	Receive By Reference
RSUM$	$DRRES	DRRES	Resume Task
RUN$	$DRRUN	DRMKT	Run Task
SCAA$	$DRCLI	DRCLI	Specify command arrival AST
SCLI$	$DRCLI	DRCLI	Set command line interpreter
SDAT$	$DRSND	DRRAS	Send Data
SDRC$	$DRSRC	DRSPW	Send, Request, and Connect
SDRP$	$DRSRC	DRSPW	Send data request and pass offspring control block
SETF$	$DRSEF	DRSED	Set Event Flag
SFPA$	$DRFEX	DRPUT	Specify Floating Point AST
SMSG$	$DRSMG	DRSMG	Send message
SPND$S	$DRSPN	DRRES	Suspend
SPRA$	$DRPUT	DRPUT	Specify Power Recovery AST
SPWN$	$DRREQ	DRSPW	Spawn
SRDA$	$DRRCV	DRPUT	Specify Receive Data AST
SREA$	$DRREX	DRPUT	Specify Requested Exit AST
SREF$	$DRSRF	DRMAP	Send By Reference
SRRA$	$DRRRA	DRPUT	Specify Receive-By-Reference AST
STLO$	$DRSTL	DRSED	Stop for Logical OR of Event Flags
STOP$S	$DRSTP	DRRES	Stop
STSE$	$DRSTS	DRSED	Stop for Single Event Flag
SVDB$	$DRSDV	DRSST	Specify SST Vector Table for Debugging
SVTK$	$DRSTV	DRSST	Specify SST Vector Table for Task
ULGF$	$DRELE	DRCMT	Unlock group global event flags
UMAP$	$DRUNM	DRMAP	Unmap Address Window
USTP$	$DRUNS	DRRES	Unstop Task
WSIG$S	$DRWSE	DRSUB	Wait for Significant Event
WTLO$	$DRWFL	DRSED	Wait for Logical OR of Event Flags
WTSE$	$DRWFS	DRSED	Wait for Single Event Flag

DEVICE DRIVER MODULES

System Device Tables (SYSTB)

● Contains DCBs, UCBs and SCBs, for devices with resident data bases.

● Data bases for standard drivers are contained in SYSTB.

● User can include other device data bases in SYSTB at SYSGEN.

RK05 Driver (DKDRV)

● Device driver for RKØ5 disk drives.

● Contains code for performing I/O to device.

● Interrupts from RKØ5 drive cause entry into DKDRV at $DKINT

Table 3-3   Standard Device Drivers

Driver	Device
DKDRV	RK05 Disk Drives
DMDRV	RK06,RK07 Disk Drives
DBDRV	RP04,RP05,RP06 Disk Drives
DRDRV	RM02,RM03,RM05,RP07,RM80 Disk Drives
DLDRV	RL01,RL02 Disk Drives
LPDRV	Line Printer
TTDRV	Terminal
MMDRV	TU16,TE16,TU45,TU77 Magnetic Tape
MTDRV	TU10,TE10,TS03,TM11 Magnetic Tape
MSDRV	TS04 Magnetic Tape
DTDRV	TC11 DECtape
DXDRV	RX01 Floppy Disk Drive
DYDRV	RX02 Floppy Disk Drive
DDDRV	TU58 Disk Cartridge
DSDRV	RS03,RS04 Disk Drives
PPDRV	Paper Tape Punch
PRDRV	Paper Tape Reader
CTDRV	TA11 Cassette Tape
DFDRV	RS08 Fixed Head Disk Drive
DPDRV	RP02 Disk Drive
LKDRV	KMC-11 Line Printer, Controller

FILE SYSTEM OVERVIEW

4

INTRODUCTION

This module introduces the FILES-11 system. FILES-11 consists of an on-disk and in-memory data structure. The on-disk is the same for a number of related operating systems (i.e., IAS, RSX-11D, etc.). The in-memory structure is RSX-11M specific. This structure is a cache of information which improves the performance of the file system.

OBJECTIVES

1. Describe the on-disk and in-memory structures of the FILES-11 system.

2. Describe the directory structure used in FILES-11.

3. Describe how files are accessed on a FILES-11 volume.

RESOURCES

1. RSX-11M/M-Plus I/O Operations Reference Manual

CONCEPTS

FILES-11

- Method of organizing data on mass-storage media

- Common to: RSX-11M/M-PLUS, RSX-11D, IAS, TRAX, VMS

File system includes:

- File processing software such as FCS or RMS

    Provides easy interface between user program and file system

    Allows access data by means of user-defined records

- Executive code to handle queued I/O requests

    Result of QIO directive

    Provides synchronization of I/O operations

    Determines if I/O operation is file-oriented

- A device driver to handle primitive I/O operations

    Data transfers between the device and memory

- An ancillary control processor, ACP, which performs file-related I/O operations

    Possible conversion to driver operations

A FILES-11 volume may be:

- Any kind of disk (including floppy disks)

- DECtape
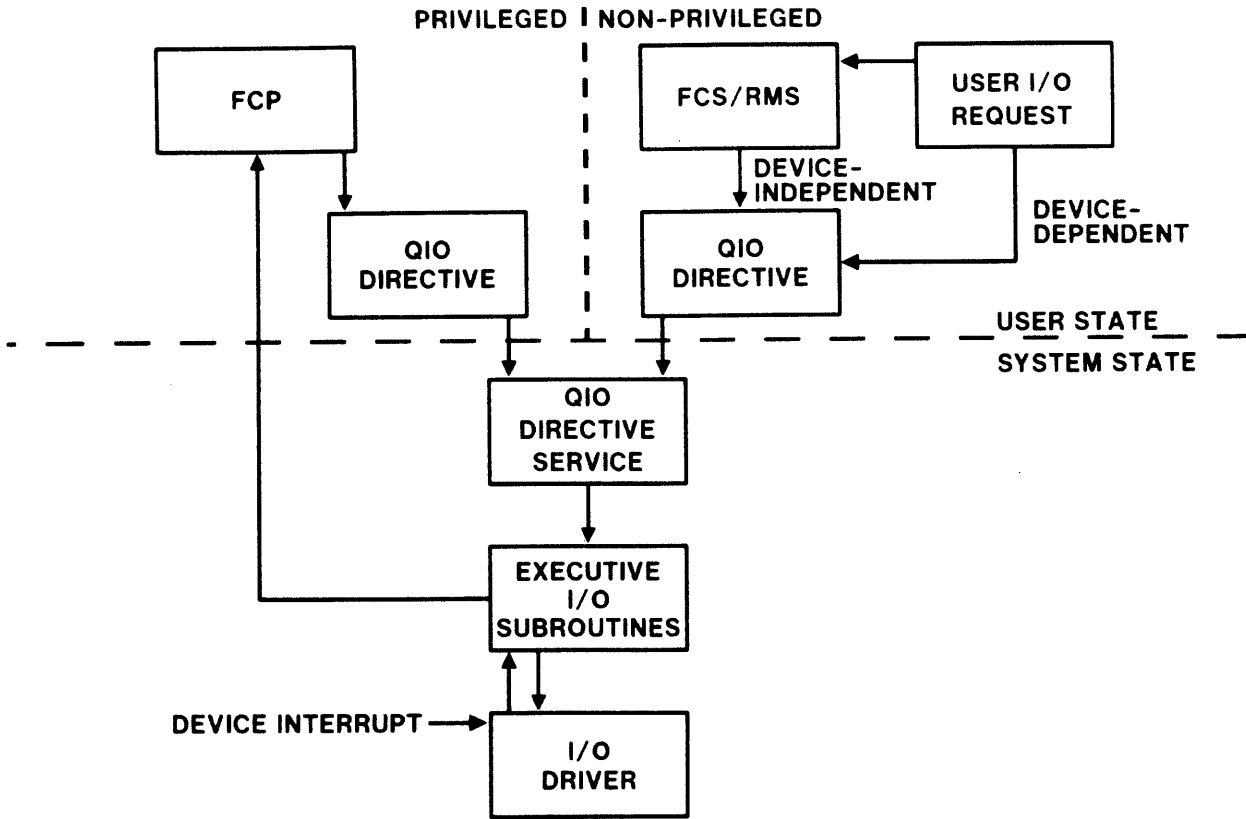
Figure 4-1   The File System

Files

Three units of measurement of space on media:

- Physical blocks

    Physically addressable unit

    Length dependent on type of media

        Usually 512 bytes
        RXØ1 - 128 bytes
        RXØ2 - 128 or 256 bytes

    Normally referenced by device driver

- Logical blocks

    Length equal to 512 bytes

    Block numbers unique for volume

        Begins at logical block Ø

        Highest logical block equals volume size minus one

    Normally referenced by the file processor (F11ACP)

- Virtual blocks

    Used by programmer to access data in files

    Length always equal to 512 bytes

    Virtual block numbers relative to file

        Begins at virtual block 1

        Highest virtual block equals file size

Files consist of two parts:

- The File Header
- The File Body

The File Header

- Contains description of file

        File name
        File owner
        File characteristics used by FCS and RMS
        Information for mapping virtual blocks to logical blocks

- Usually one block long (512. bytes); size depends on the file processor in use

The File Body

- Contains data stored by user and any control data appended by the file access method

- Consists of a number of contiguous groups of logical blocks (segments)

FILE HEADER BLOCK

(DISK
BLOCK
256 WORDS)

| OFFSETS TO IDENTIFICATION AND MAP AREAS
FILE ID
FILE OWNERSHIP AND PROTECTION INFORMATION
FILE CHARACTERISTICS
SIZE (BYTES) OF HEADER AREA | HEADER AREA |

OFFSETS TO IDENTIFICATION
   AND MAP AREAS
FILE ID
FILE OWNERSHIP AND PROTECTION
   INFORMATION
FILE CHARACTERISTICS
SIZE (BYTES) OF HEADER AREA

HEADER AREA

FILE NAME, TYPE AND VERSION

DATES OF CREATION AND
   REVISION

SIZE (BYTES) OF IDENTIFICATION
   AREA

IDENTIFICATION AREA

MAPPING INFORMATION

RETRIEVAL POINTERS

CHECKSUM WORD

MAP AREA

Figure 4-2   The File Header

```
Dump of DB2:[305,306]INTRO1.MOD;26 - File ID 7450,60,0
                                        File header

HEADER AREA
        H.IDOF          027
        H.MPOF          056
        H.FNUM,
        H.FSEQ          (7450,60)
        H.FLEV          401
        H.FOWN          [305,306]
        H.FPRO          [RWED,RWED,RWED,R]
        H.UCHA          000 =
        H.SCHA          000 =
        H.UFAT
                F.RTYP  002 = R.VAR
                F.RATT  002 =    FD.CR
                F.RSIZ  102 = 66.
                F.HIBK  H:0 L:000013 = 11.
                F.EFBK  H:0 L:000013 = 11.
                F.FFBY  310 = 200.
                (REST)
                000000 000000 000000 000000 000000 000000 000000 000000
                000000
IDENTIFICATION AREA
        I.FNAM,
        I.FTYP,
        I.FVER          INTRO1    .MOD;26
        I.RVNO          6
        I.RVDT          14-DEC-81
        I.RVTI          02:50:22
        I.CRDT          15-OCT-81
        I.CRTI          09:46:39
        I.EXDT          --
MAP AREA
        M.ESQN          000
        M.ERVN          000
        M.EFNU,
        M.EFSQ          (0,0)
        M.CTSZ          001
        M.LBSZ          003
        M.USE           002 = 2.
        M.MAX           314 = 204.
        M.RTRV
        SIZE    LBN
        11.     H:001 L:165751 = 125929.
CHECKSUM
        H.CKSM          123624
```

Example 4-1   The File Header for a Data File

```
Dump of DB2:[305,306]SPY.TSK;16 - File ID 16104,7,0
                                         File header

HEADER AREA
        H.IDOF          027
        H.MPOF          056
        H.FNUM,
        H.FSEQ          (16104,7)
        H.FLEV          401
        H.FOWN          [305,306]
        H.FPRO          [RWED,RWED,RWED,R]
        H.UCHA          200  =   UC.CON
        H.SCHA          000  =
        H.UFAT
              F.RTYP    001  =  R.FIX
              F.RATT    000  =
              F.RSIZ    1000 =  512.
              F.HIBK    H:0 L:000050 = 40.
              F.EFBK    H:0 L:000051 = 41.
              F.FFBY    0 = 0.
              (REST)
              000000 000000 000000 000000 000000 000000 000000 000000
              000000
IDENTIFICATION AREA
        I.FNAM,
        I.FTYP,
        I.FVER          SPY          .TSK;16
        I.RVNO          22
        I.RVDT          14-DEC-81
        I.RVTI          02:50:28
        I.CRDT          05-JUN-81
        I.CRTI          14:14:33
        I.EXDT          ---
MAP AREA
        M.ESQN          000
        M.ERVN          000
        M.EFNU,
        M.EFSQ          (0,0)
        M.CTSZ          001
        M.LBSZ          003
        M.USE           002 = 2.
        M.MAX           314 = 204.
        M.RTRV
        SIZE      LBN
        40.       H:001 L:172544 = 128356.
CHECKSUM
        H.CKSM          133671
```

Example 4-2 The File Header for a Task Image

Retrieval Pointers

- Defines the correspondence between logical and virtual blocks

- Contained in the file header:

    Maximum: 102 per header

    One fragment per retrieval pointer

    Maximum file length with 1 block per header: 26112 blocks

- Four bytes long

    One byte: number of virtual blocks in segment; maximum is 256 blocks.

    Three bytes: pointer to first logical block in segment.

```
┌─────────────────────┬──────────────┐
│ NO. BLOCKS - 1      │              │
├─────────────────────┴──────────────┤
│          STARTING LBN               │
└─────────────────────────────────────┘
```

(1,3) RETRIEVAL POINTER
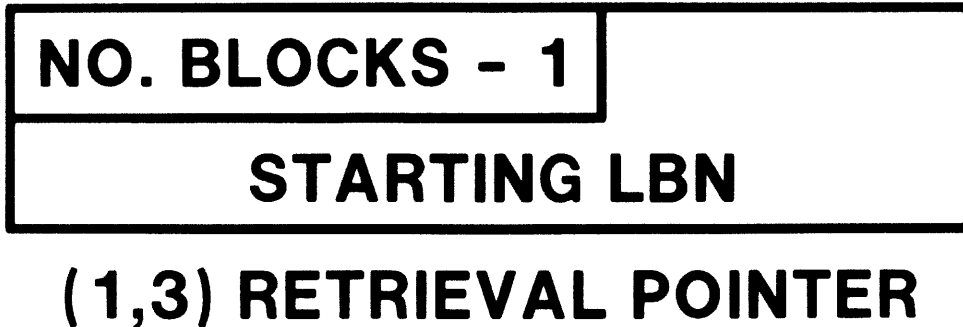
Figure 4-3 Retrieval Pointer

FILE HEADER

FILE BODY

VBN   LBN

0	
406	
1	
301	
0	
701	

RETRIEVAL
POINTERS

2   301

3   302

1   406

4   701
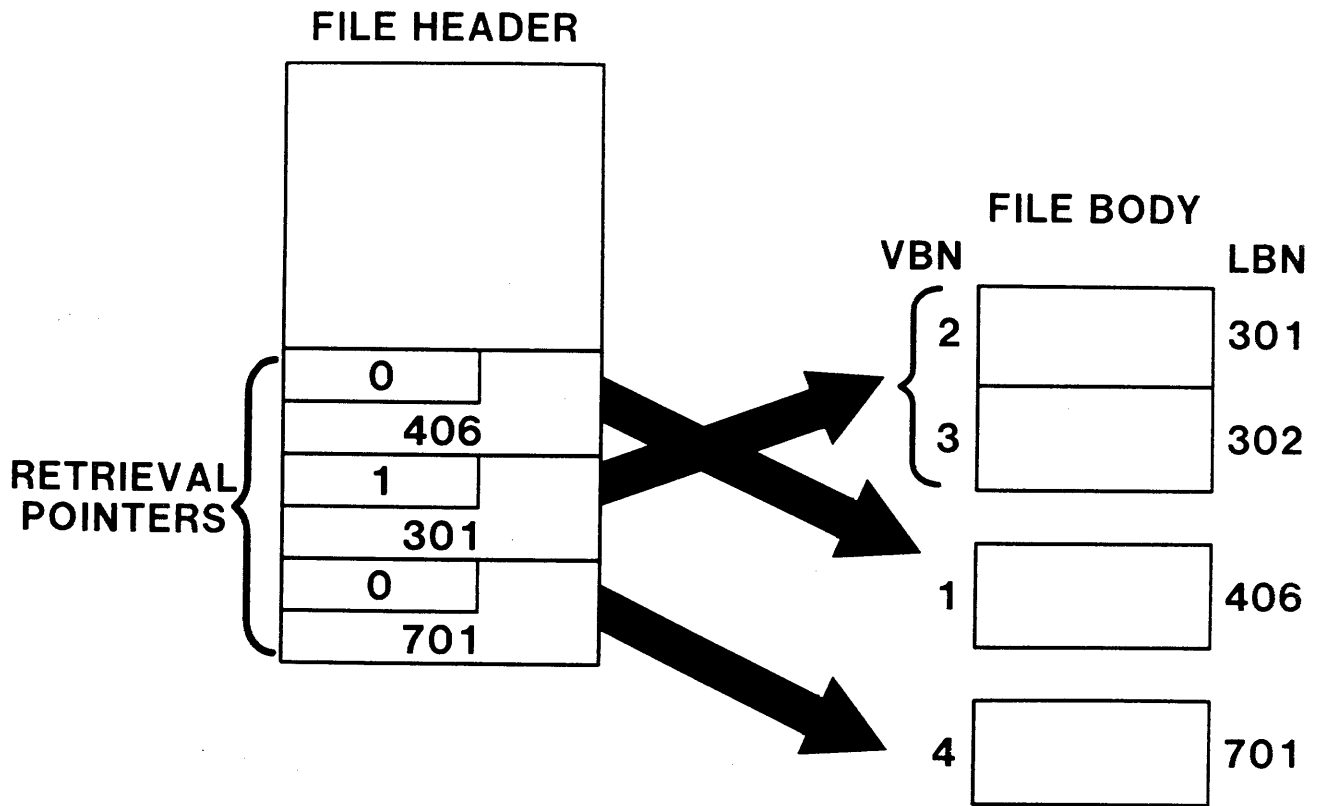
Figure 4-4   Mapping Virtual to Logical Blocks

File Directories

● Contains information needed to identify and locate files

File name, type and version number

File number (Relative number of file headers in index file)

File sequence number (Number of times a file header is used)

● Two types of directories

Master File Directory (MFD) -- One per volume

User File Directory (UFD) -- As many as desired per volume

● Naming convention

Name of directory: xxxyyy.DIR;1

where

xxx is a group number
yyy is a member number

Master File Directory: 000000.DIR;1

User File Directory for UIC: [6,32]: 006032.DIR;1

1	**FILE ID**
2	**FILE SEQUENCE NO.**
3	**0**
4	
5	**FILENAME**
6	**(RAD50)**
7	**FILE TYPE (RAD50)**
8	**FILE VERSION**

Figure 4-5 Directory Record

80

FILES-11 VOLUMES

The Standard Files

- The Index File: INDEXF.SYS

- The Bitmap File: BITMAP.SYS

- The Master File Directory: 000000.DIR

- The Bad Block File: BADBLK.SYS

- The System Checkpoint File: CORIMG.SYS

The Index File

- Contains basic structures needed by file system.

      Boot Block
      Home Block
      Index File Bitmap
      All File Headers

- Filespec: [0,0]INDEXF.SYS;1,
  File ID/SEQ: 1/1

- Boot Block

      Contains bootstrap code for hardware booting of device

      Always virtual block 1 of index file

      Always logical block 0 of volume

      Written by SAV command when the /WB switch is used

      If volume not bootable, contains code to print the
      message: THIS VOLUME DOES NOT CONTAIN A HARDWARE BOOTABLE
      SYSTEM

- Home Block

      Contains information needed by file system to access data
      on volume

      Volume label

      Pointer to index file bitmap

      Size of index file bitmap

      Checksum words to validate data in home block

      Always located in virtual block 2 of index file

      Located in first good block in sequence
      1, 400, 1000, 1400, ...

      Checksum used to identify homeblock

FILE SYSTEM OVERVIEW

- Index File Bitmap

    Allows fast access to unused blocks in index file

    Always located beginning in virtual block 3 of index file

    One bit per possible file header in index file

    Bit is set to indicate if block is in use

    Bit clear if block is free

    Bit is set if block is in use

- File Headers

    Contained in virtual blocks starting at block 3+n, where n
    is the length of the bitmap

    First 16 headers are contiguous

    Virtual block of header = 2 + m + n

    where

    n = number of blocks in index file bitmap
    m = file number

VBN		LBN
1	BOOT BLOCK	0

		1 OR 400 OR 1000...
2	HOME BLOCK	

| 3 | INDEX BIT MAP
0=FREE
1=USED | |

3+N	INDEX FILE HEADER	N IS SIZE OF INDEX BITMAP (NO. BLOCKS)
4+N	STORAGE BIT MAP FILE HEADER	
5+N	BAD BLOCK FILE HEADER	
6+N	MASTER FILE DIRECTORY FILE HEADER	
7+N	CORE IMAGE CHECKPOINT FILE HEADER	

| 8+N | USER FILE DIRECTORY
FILE HEADER | |
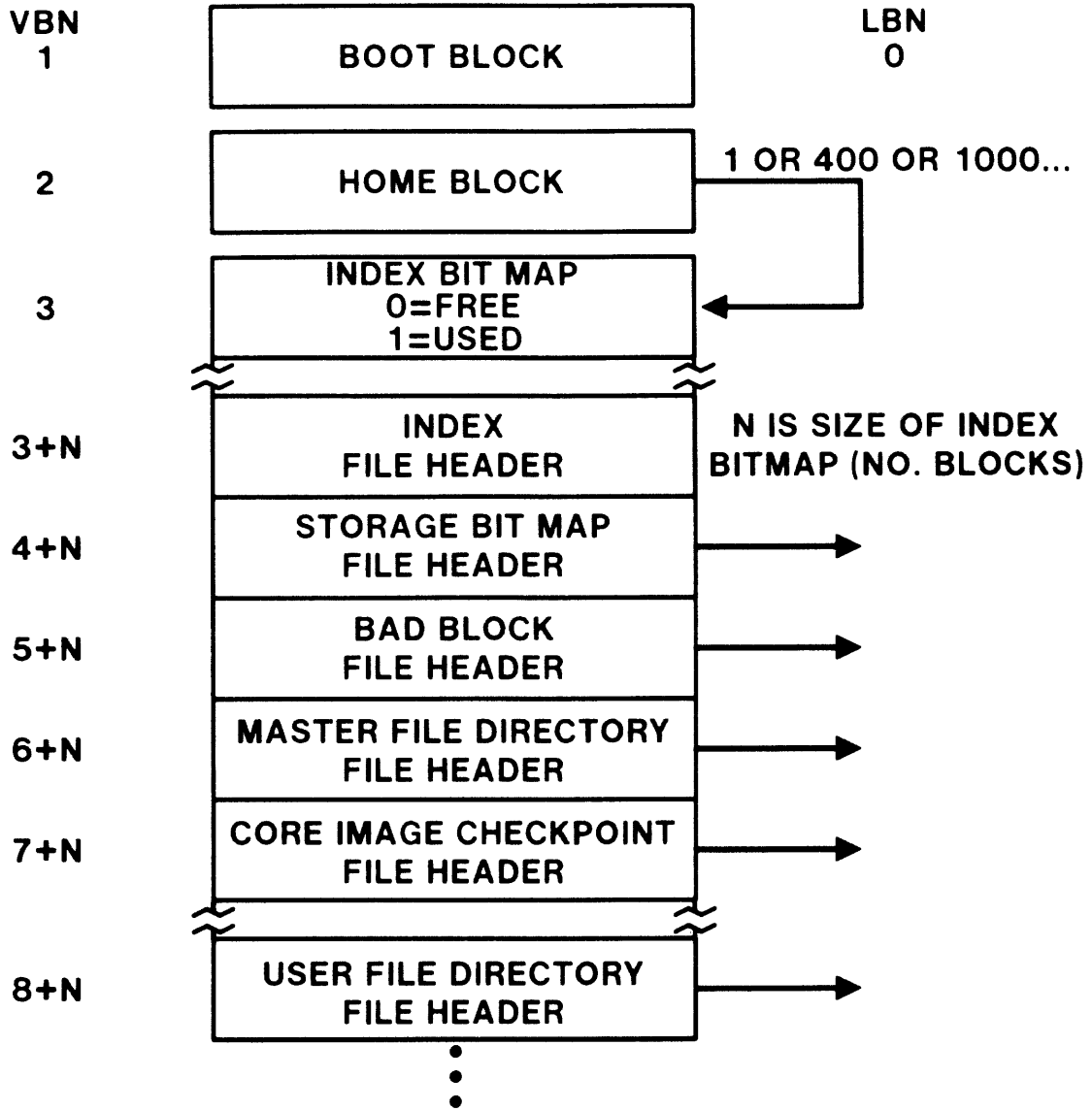
Figure 4-6 Structure of the Index File

84

The Bitmap File

- Virtual block number 1 contains allocation information used on RSX-11D

- Bitmap starts in virtual block number 2 of the bitmap file

- One bit per logical block on volume

- Value of each bit indicates status of corresponding block

    Bit set - block not allocated to file

    Bit clear - block allocated

- VFY compares clear bits with retrieval pointers

- Filespec: [0,0]BITMAP.SYS;1,
  File ID/SEQ: 2/2
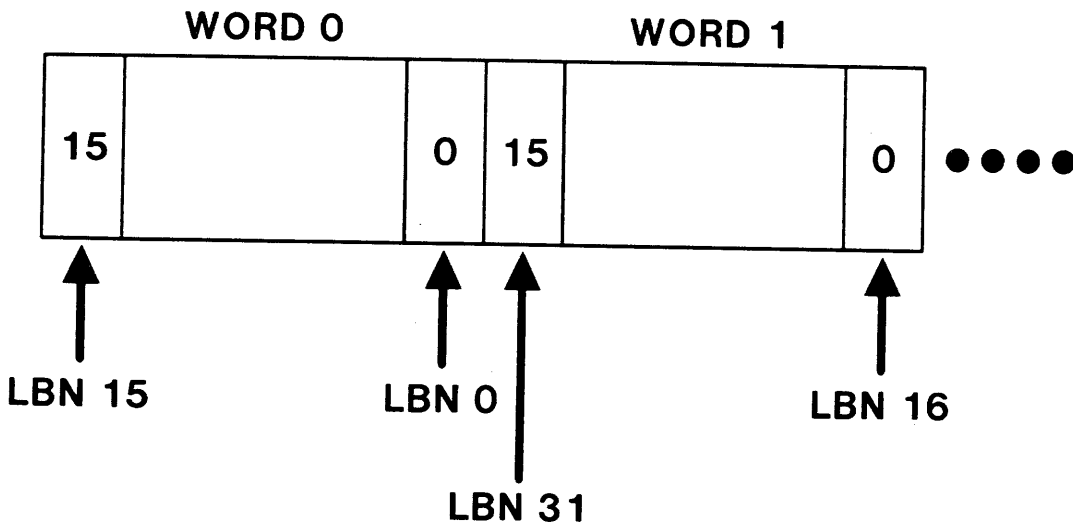
Figure 4-7 The Bitmap File

The Master File Directory

● Directory corresponding to UFD [Ø,Ø]

● Contains entries for the five standard files

● Contains entries for all UFDs on volume

● Contains entry for system account file RSX11.SYS

● Filespec: [Ø,Ø]ØØØØØØ.DIR;1,
  File ID/SEQ: 4/4

The Badblock File

● Used to keep bad blocks from being allocated to user files

● Bad block information created by BAD or entered by the user or
  manufacturer allocates blocks to file

● Filespec: [Ø,Ø]BADBLK.SYS;1, File ID/SEQ: 3/3

● No dynamic badblock handling

```
                              ┌───────┐
                              │  MFD  │
                              └───┬───┘
   ┌──────┬──────┬──────┬──────┬──────┬──────┐
┌──┴──┐┌──┴──┐┌──┴──┐┌──┴──┐┌──┴──┐┌──┴──┐┌──┴──┐
│INDEXF││BITMAP││BADBLK││000000││CORIMG││GGGMMM││XXXXXX│ ●●●
│ SYS  ││ SYS  ││ SYS  ││ DIR  ││ SYS  ││ DIR  ││ DIR  │
└──────┘└──────┘└──────┘└──────┘└──────┘└──────┘└──┬───┘
                         ┌──────────┬──────────┐
                     ┌───┴────┐┌────┴───┐┌─────┴──┐
                     │FILE MAC││FILE OBJ││FILE TSK│
                     └────────┘└────────┘└────────┘
```
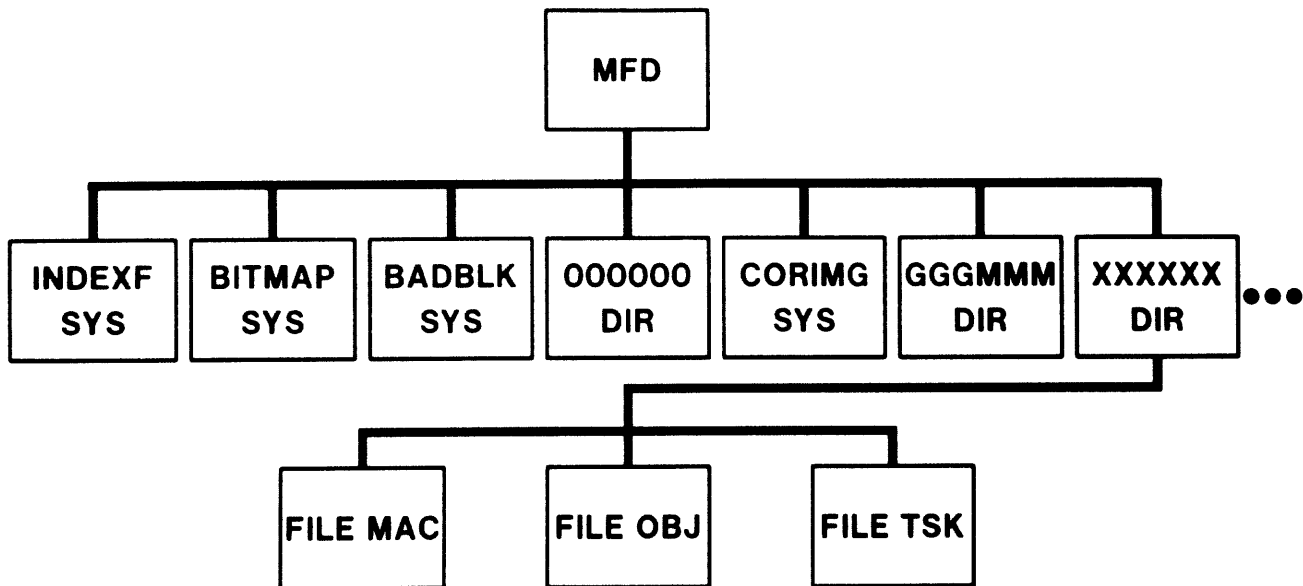
Figure 4-8 Two Level Directory Structure

86

The System Checkpoint File

- Used to checkpoint tasks

- No space allocated when volume initialized

- Space allocated by ACS command

- Always contiguous

- Filespec: [0,0]CORIMG.SYS;1,
  File ID/SEQ: 5/5

FUNCTIONS OF THE FILE SYSTEM

Creating a FILES-11 Volume

- FILES-11 volume created by INITIALIZE command

- Home block information can be modified by the HOME command

- Creates five standard files

- Volume characteristics determined:

    Volume label

    Number of blocks for default file extension

    Size of index file (number of blocks used for file headers)

    Number of files on volume

    Location of index file

    Default protection for files on volume

Creating a File

- Index file bitmap searched for first free block

- First free block used for file header

- Bitmap file searched for required number of free blocks to allocate initial file

- Entry for new file placed in UFD

```
┌──────────────┐      ┌──────────────┐      ┌──────────────┐      ┌──────────────┐
│   CREATE     │      │   CREATE     │      │  ALLOCATE    │      │ MAP VIRTUAL  │
│    FILE      │ ───▶ │  DIRECTORY   │ ───▶ │  BLOCKS TO   │ ───▶ │ TO LOGICAL   │
│   HEADER     │      │   ENTRY      │      │  FILE BODY   │      │   BLOCKS     │
└──────────────┘      └──────────────┘      └──────────────┘      └──────────────┘
        │                    │                     │                     │
        ▼                    ▼                     ▼                     ▼
┌──────────────┐      ┌──────────────┐      ┌──────────────┐      ┌──────────────┐
│  USE INDEX   │      │  ENTER FILE  │      │ USE BITMAP   │      │              │
│ FILE BITMAP  │      │  NAME INTO   │      │  FILE TO     │      │  INITIALIZE  │
│ TO ALLOCATE  │      │    UFD       │      │LOCATE UNUSED │      │  RETRIEVAL   │
│   HEADER     │      │              │      │  BLOCKS      │      │  POINTERS    │
└──────────────┘      └──────────────┘      └──────────────┘      └──────────────┘
        │                    │
        ▼                    ▼
┌──────────────┐      ┌──────────────┐
│  INITIALIZE  │      │  SET UP FILE │
│ FILE HEADER  │      │    ID IN     │
│              │      │   HEADER     │
└──────────────┘      └──────────────┘
```
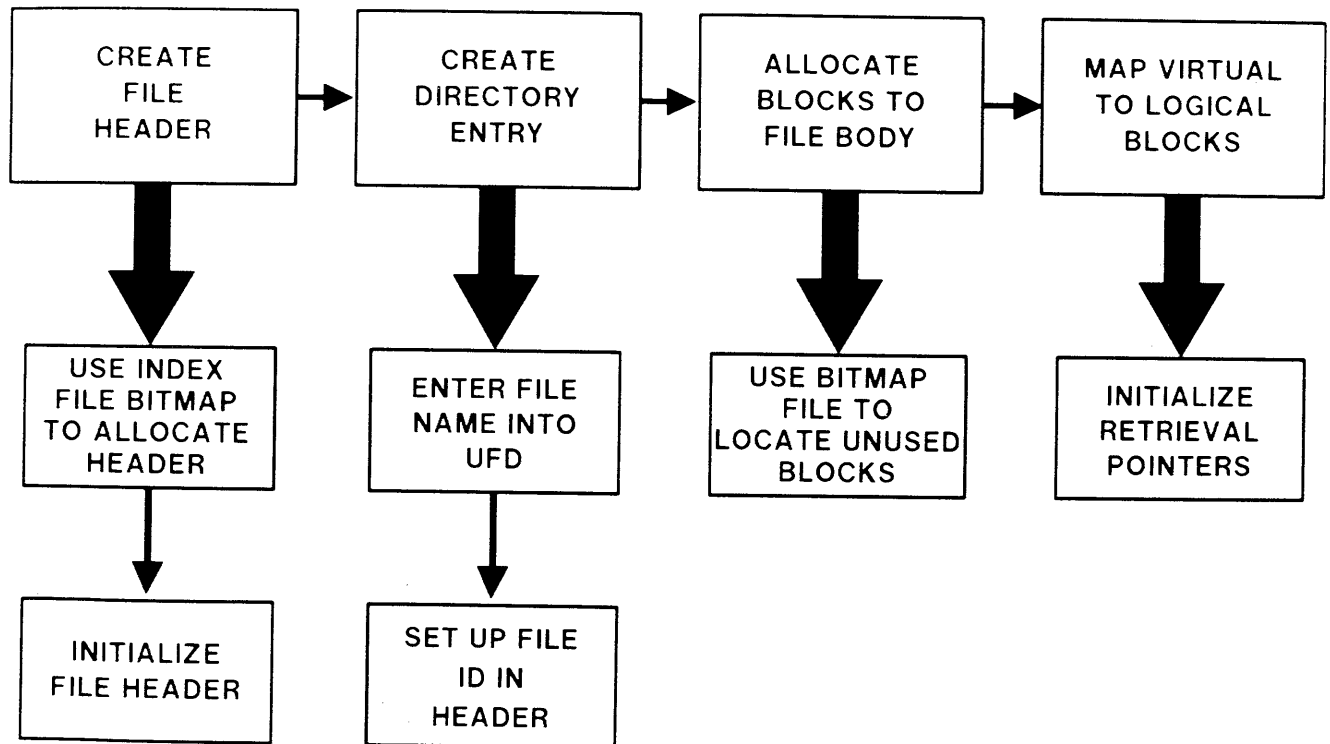
Figure 4-9 Creating a File

Accessing a File

● By file name - DB1:[101,101]DATA.DAT;1

MFD from device mounted on DB1: is searched for 101101.DIR

File ID for 101101.DIR is used to find the file header

Retrieval pointers from header are used find UFD file

UFD is searched for entry for DATA.DAT

File ID for DATA.DAT is used to find the file header

Retrieval pointers from header are used to find file

File is accessed

● If file ID is known, file header can be accessed without using directories

INDEX FILE

VBN

4	INDEX FILE HEADER
5	BITMAP FILE HEADER
6	BADBLOCK FILE HEADER
7	MFD FILE HEADER
10	CORIMG FILE HEADER
11	UFD [1,1] HEADER
12	UFD [200,200] HEADER
13	UFD [101,101] HEADER
14	DATA.DAT;1 HEADER

MFD ID,SEQ,0

INDEXF.SYS;1	1,1,0
BITMAP.SYS;1	2,2,0
BADBLK.SYS;1	3,3,0
000000.DIR;1	4,4,0
CORIMG.DIR;1	5,5,0
001001.DIR;1	6,1,0
200200.DIR;1	7,1,0
005006.DIR;1	0,3,0
101101.DIR;1	10,6,0

UFD [101,101]

A.FTN;1	0,5,0
DATA.DAT;1	11,1,0

DATA.DAT;1

USER FILE DATA

Figure 4-10 Accessing a File

THE FILE PROCESSOR

File Control Processors

- Privileged system task which handles file-specific I/O

- Processes QIO commands queued to its receive queue

- Several file processors supplied with system

  F11ACP - Used for FILES-11 disk volumes

  MTAACP - Used for magnetic tape volumes

  NETACP - Used by DECNET

- To speed up file operations, dedicate appropriate FCPxxn to volumes

  >INS $FCPxxx/TASK=DB1FCP

  >MOU DK∅:/ACP=DB1FCP

Logical Units

- Tasks specify I/O devices by using logical unit numbers.

- Each logical unit number refers to a unique I/O device.

- Pointer to an entry of the logical unit table for the task.

- The logical unit table is in the task header.

- Each entry is two words long:

    Word one is a pointer to the UCB for the device.

    Word two is a pointer to a FILES-11 window block if file
    is open; and Ø otherwise.

- FILES-11 window block is a data structure used to store
  retrieval pointers.

- Logical unit table constructed and units assigned at
  taskbuild.

- Logical unit assignments can be changed by:

    REASSIGN command

    ALUN$ directive

TASK HEADER

NUMBER OF LUNs = 6

LUN 1 { POINTER TO UCB / POINTER TO WINDOW BLOCK IF FILE OPEN

LUN 2 { POINTER TO UCB / POINTER TO WINDOW BLOCK IF FILE OPEN

LUN 3 {

LUN 4 {

LUN 5 {

LUN 6 {
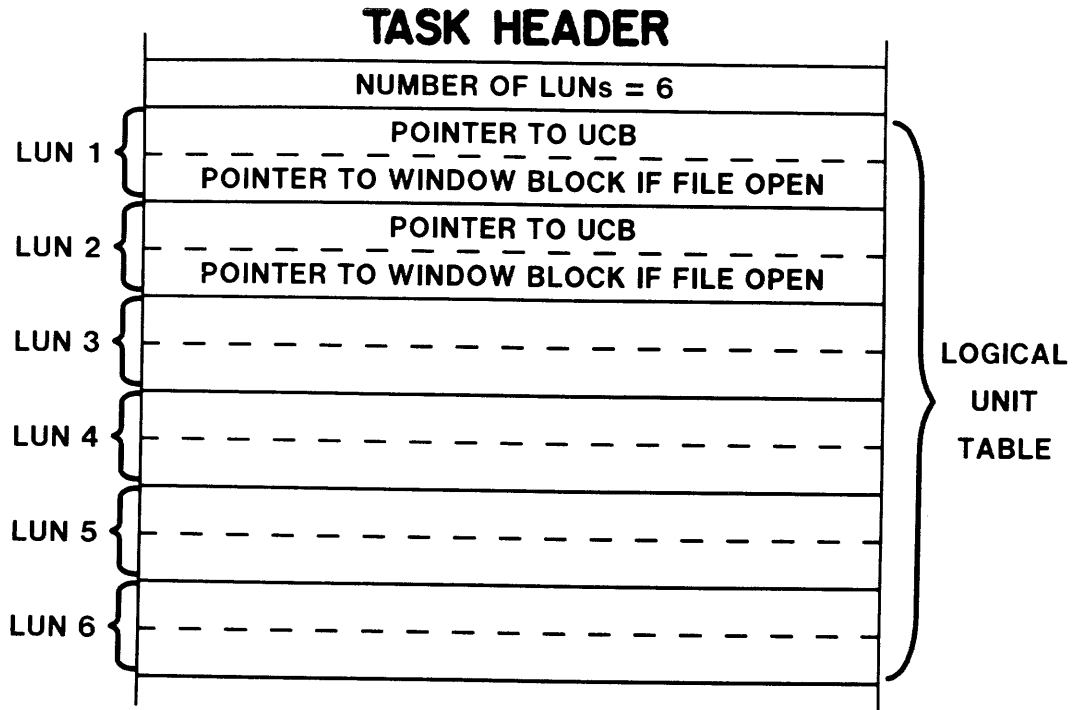
LOGICAL UNIT TABLE

Figure 4-11   The Logical Unit Table

Data Structures

- The Volume Control Block (VCB)

    Created from information in home block

    One VCB per mounted volume

    Contains

        Transaction count for volume
        Logical block number of index file bitmap
        Logical block number of storage bitmap
        Pointers to other data structures, e.g.   FCB

- The File Control Block (FCB)

    Created from file header

    One FCB per open file

    Contains

        Logical block number of file header
        File number and sequence numbers
        File owner and protection code

- FILES-11 Window Block

    One window block per accessor to file

    Contains

        Access privileges of accessor
        Retrieval pointers (seven by default)

NOTE

Retrieval pointers in the file header use one
byte for the block count and three bytes for
the disk address; those in the window block
use two bytes for the count and four bytes
for the disk address.

94

VOLUME TRANSACTION COUNT		V.TRCT
VOL. CHARACTERISTICS	VOLUME TYPE	V.TYPE
VOLUME LABEL (ASCII) (SEVEN WORDS)		V.LABL
PACK SERIAL NUMBER (FOR ERROR LOGGING)		V.PKSR
INDEX FILE WINDOW ADDRESS		V.IFWI
FILE CONTROL BLOCK LISTHEAD		V.FCB

V.VCHA (labels the VOL. CHARACTERISTICS / VOLUME TYPE row)
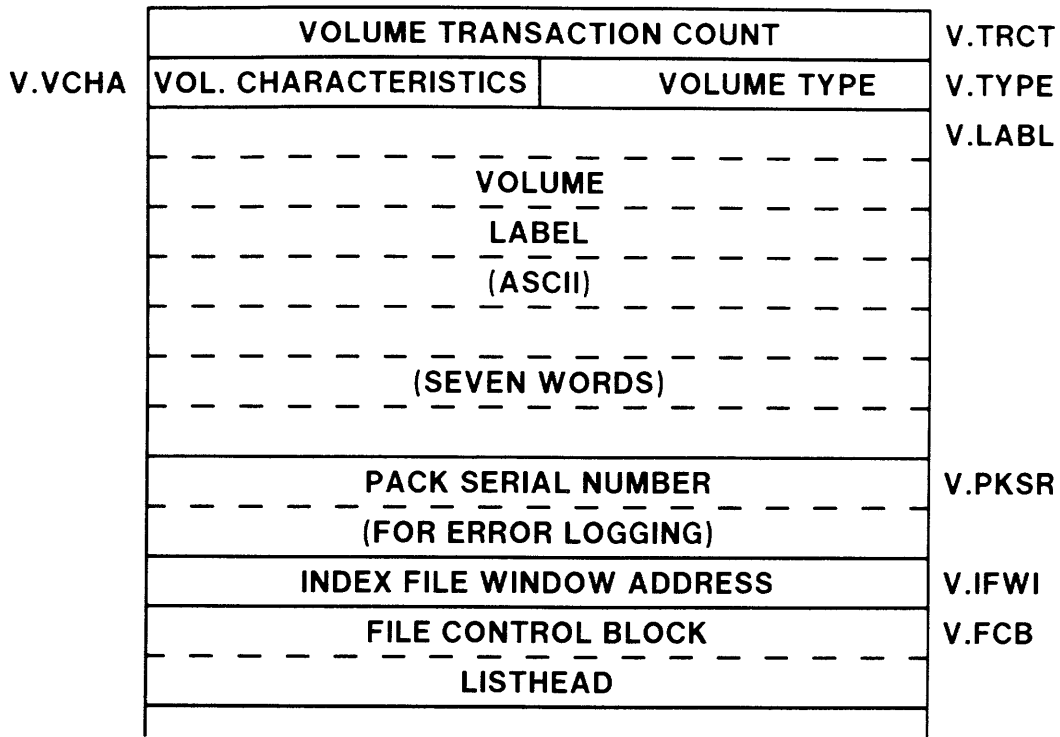
Figure 4-12   The Volume Control Block
(Part One)

V.IBSZ	INDEX FILE BITMAP SIZE	_ _ _ _ _ _ _	V.IBLB
	INDEX FILE BITMAP LBN		
	MAXIMUM NUMBER OF FILES		V.FMAX
V.SBCL	BITMAP FILE CLUSTER	DEFAULT SIZE OF WINDOW	V.WISZ
	BITMAP FILE SIZE		V.SBSZ
V.FIEX	DEFAULT FILE EXTENSION	_ _ _ _ _ _ _	V.SBLB
	BITMAP FILE LBN		
	VOLUME OWNER UIC		V.VOWN
	VOLUME PROTECTION WORD		V.VPRO
	DEFAULT FILE PROTECTION WORD		V.FPRO
V.LRUC	AVAILABLE LRU SLOTS	_ _ _ _ _ _ _	V.FRBK
	NUMBER OF FREE BLOCKS ON VOLUME		
V.FFNU	1ST FREE BLOCK IN INDEX BITMAP	VOLUME STATUS	V.STS
	RESERVED FOR FUTURE USE		V.EXT

Figure 4-12   The Volume Control Block
(Part Two)

	POINTER TO NEXT FCB	F.LINK
	FILE NUMBER	F.FNUM
	FILE SEQUENCE NUMBER	F.FSEQ
F.FSQN	FILE SEGMENT / UNUSED	
	UIC OF FILE OWNER	F.FOWN
	FILE PROTECTION CODE	F.FPRD
F.SCHA	SYSTEM CHARACTERISTICS / USER CHARACTERISTICS	F.UCHA
	FILE HEADER	F.HDLB
	LOGICAL BLOCK NUMBER	
	LOGICAL BLOCK NUMBER OF VIRTUAL BLOCK 1	F.LBN
	OF CONTIGUOUS FILE (0 OTHERWISE)	

F.NLCK	SIZE OF FILE	F.SIZE
	IN BLOCKS	
	NUMBER OF LOCKS / NUMBER OF ACCESSES	F.NACS
	STATUS BITS / NUMBER OF WRITE ACCESSES	F.STAT
	DIRECTORY EOF BLOCK NUMBER	F.DREF
	1ST WORD OF DIRECTORY NAME	F.DRNM
	POINTER TO EXTENSION FCB	F.FEXT
	STARTING VIRTUAL BLOCK NUMBER	F.FVBN
	OF THIS FILE SEGMENT	
	POINTER TO LOCKED BLOCK LIST	F.LKL
	WINDOW BLOCK LIST	F.WIN

Figure 4-13   The File Control Block

CONTROL BITS	NUMBER OF ACTIVE MAP ENTRIES	W.CTL
RESERVED	I/O COUNT	W.IOC
FCB ADDRESS		W.FCB
POINTER TO LIST OF LOCKED BLOCKS		W.LKL
WINDOW BLOCK LIST LINKWORD		W.WIN
SIZE OF WINDOW	1ST VIRTUAL	W.VBN
BLOCK MAPPED BY WINDOW		
NUMBER OF BLOCKS MAPPED - 1		W.RTRV
1ST PHYSICAL BLOCK		
MAPPED BY POINTER		
NUMBER OF BLOCKS MAPPED - 1		
1ST PHYSICAL BLOCK		
MAPPED BY POINTER		

W.WISZ

1ST RETRIEVAL POINTER

2ND RETRIEVAL POINTER
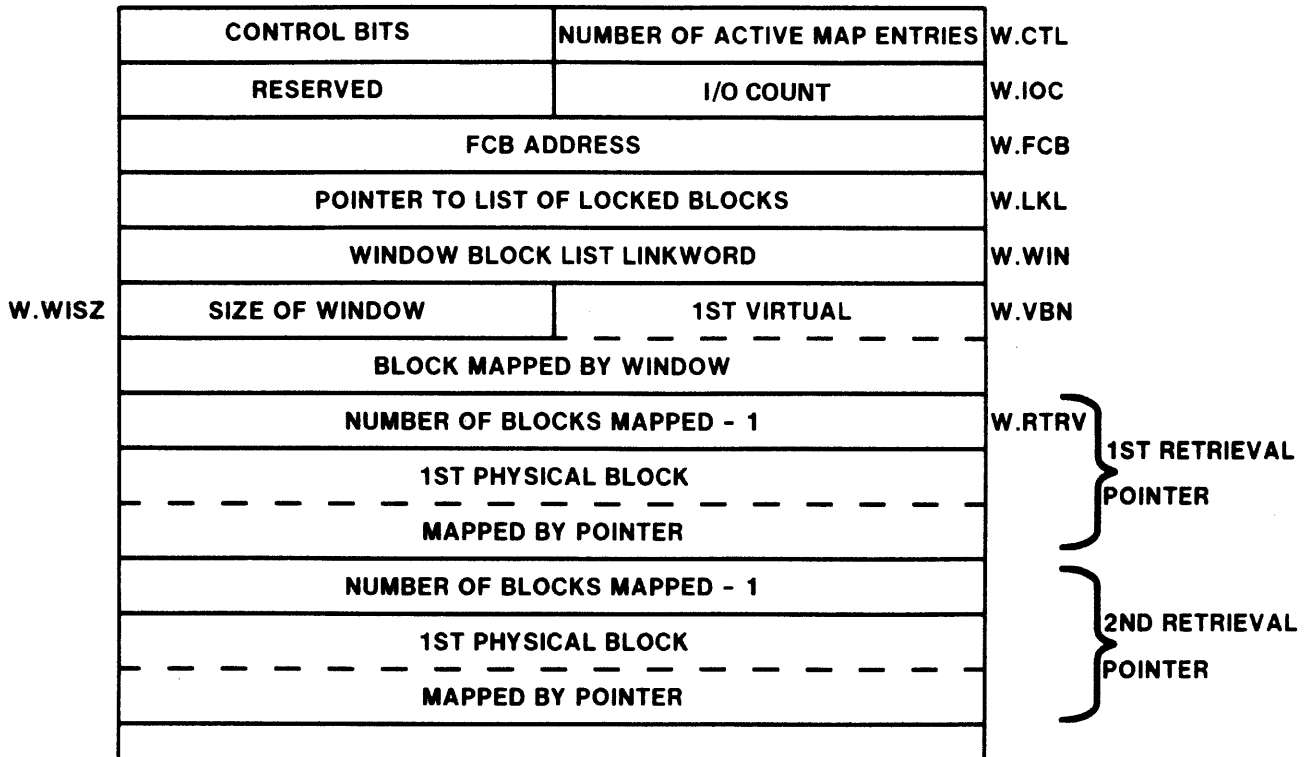
Figure 4-14   FILES-11 Window Block

98

Mounting a Volume

- Volume mounted by MOUNT command

    Compares the volume label with that specified in the MOUNT command

    Copies homeblock information into VCB

    Associates an ACP with the volume

- Steps in mounting a FILES-11 volume:

    Volume Control Block is created

    VCB is linked from the UCB using U.VCB

    Index file is opened
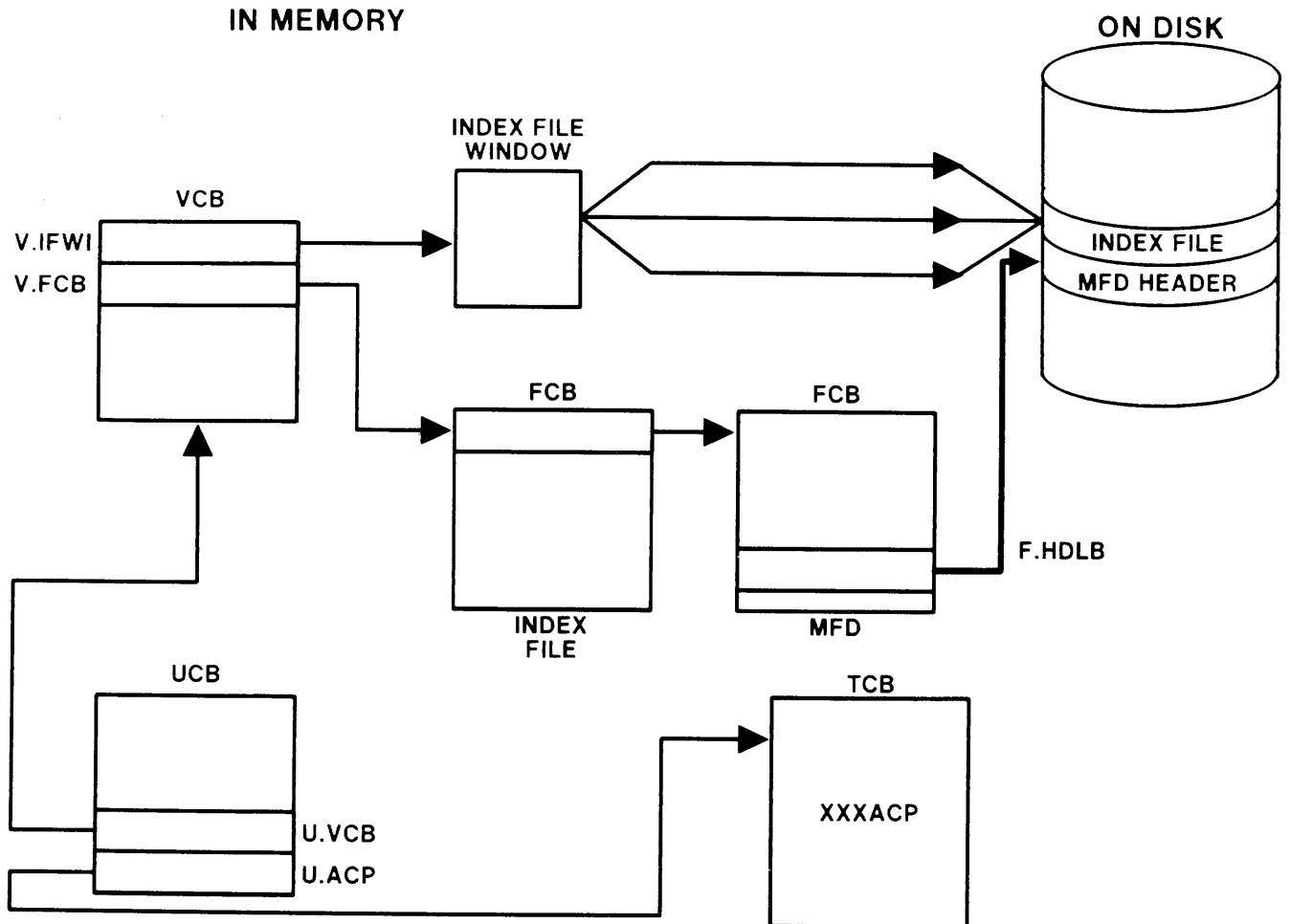
Figure 4-15 Mounting a Volume

Opening a File

- Steps in opening a file:

    File Control Block or FCB is created.

    FILES-11 window block is created for file.

    FCB is inserted in the FCB queue for volume.

    Pointer from logical unit table to window block is initialized.

- Shared files have only one FCB but separate window blocks.

- File system keeps, in memory, a list of FCBs of the more recently used directories.

    This list is called the LRU.

    Eliminates need to read directories from disk when file is accessed.

    Least recently used directory is flushed when new directory accessed if list is full.

    Size of LRU list is determined by LRU switch on INI or HOM command.
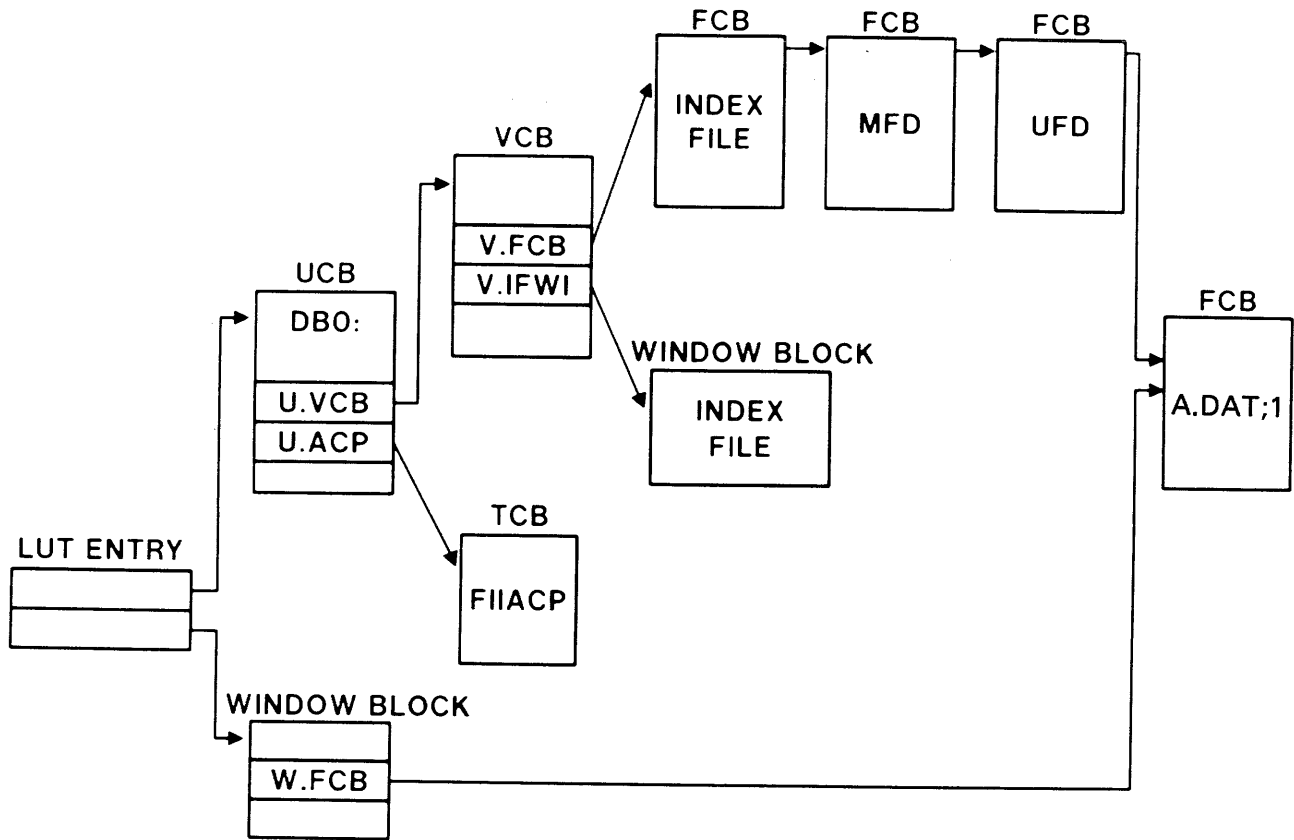
Figure 4-16 Opening a File
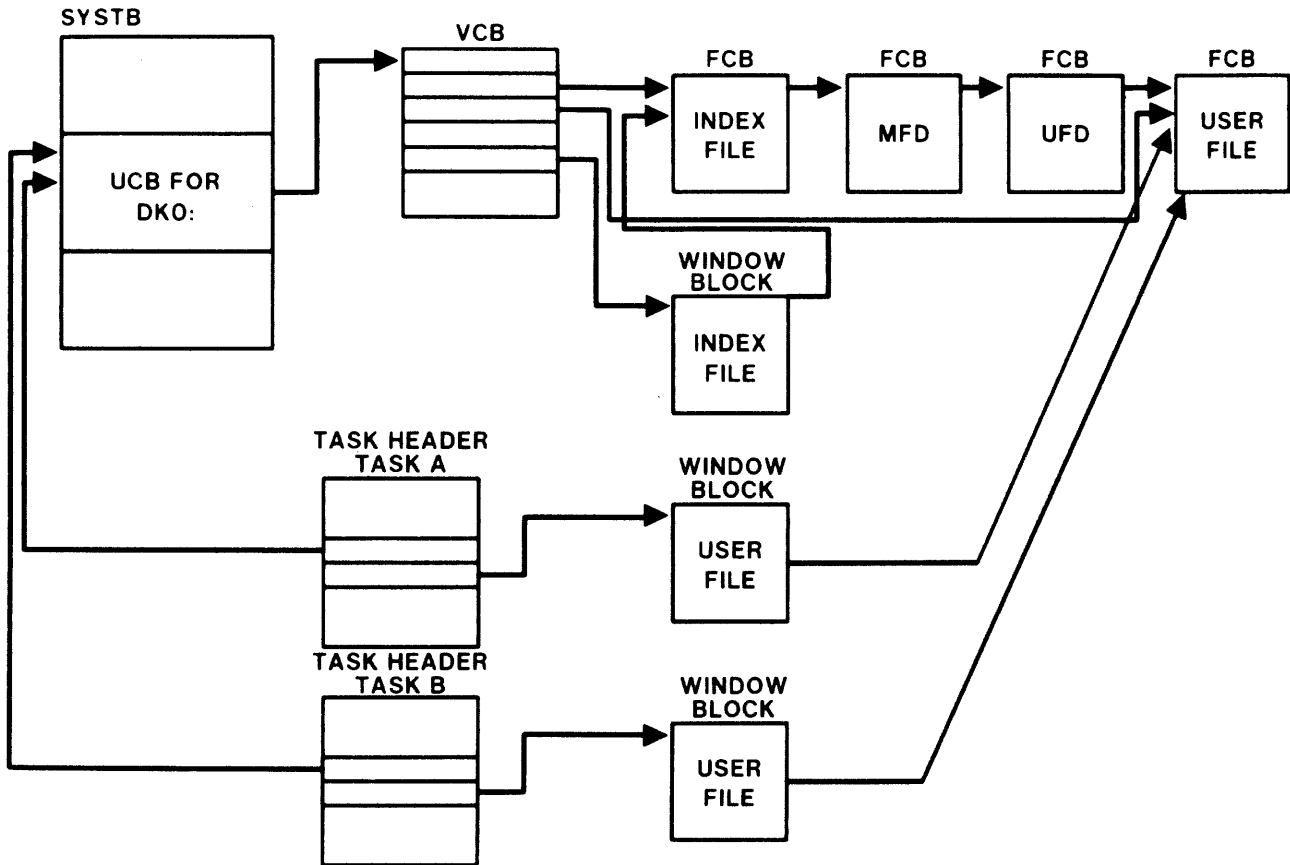
Figure 4-17   Shared Access

SYSTEM GENERATION

5

INTRODUCTION

The RSX-11M operating system is designed to support a large variety of application environments and possible hardware configurations available to the PDP-11 processor. The process used to adapt the system to these varied conditions is called System Generation or SYSGEN.

The RSX-11M SYSGEN procedure is designed to allow customizing of the operating system to be performed in an efficient and accurate manner. The procedure is run under the control of supplied indirect command files. Facilities such as VMR and SAV are provided to help the user set up his system quickly.

OBJECTIVES

1. Generate a functioning RSX-11M system.

2. Perform the ancillary functions, VMR and SAV, to produce a practical system.

RESOURCES

1. RSX-11M System Generation and Management Guide

OVERVIEW

SYSGEN Functions

- Provide support for existing hardware

    The processor model (e.g., PDP-11/44)

    Processor options (e.g., memory management, floating-point processor)

    Devices installed on system

    Number of controllers and physical units

    Vector and CSR addresses for all devices

- Provide needed software features

    Executive options (e.g., directives, 'round-robin')

    File system (version of F11ACP)

    Resident library support (e.g., FCPRES)

    Terminal support (e.g., full-duplex)

The SYSGEN Process

● SYSGEN procedure is contained in three indirect command files

    SYSGEN.CMD - First phase of SYSGEN

        Assembles Executive
        Assembles device drivers
        Creates command files to be used in phase two

    SYSGEN2.CMD - Second phase of SYSGEN

        Taskbuilds the Executive
        Taskbuilds the device drivers
        Creates needed libraries
        Taskbuilds the privileged system tasks

    SYSGEN3.CMD - Third phase of SYSGEN

        Allows system tasks to be rebuilt

● Between phase two and phase three, VMR should be applied to the new system

    Use VMR command file SYSVMR.CMD created by the SYSGEN procedure

        Can be edited to add desired partitions
        Must be edited if system is unmapped

    Configure the new system

        Set up partitions
        Load device drivers
        Install system tasks

● SYSGEN procedure creates saved answer file SYSSAVED.DAT

    Contains answers to SYSGEN questions in symbolic form
    File can be modified by user for use in later SYSGEN

Types of SYSGEN

- SYSGEN can be performed using the base-line system on the distribution kit

    Called a standalone SYSGEN

    System is dedicated to SYSGEN, no other users

- SYSGEN can be performed using an existing RSX-11M system

    Called an on-line SYSGEN

    Other users can use the system during most of the SYSGEN process

- MCR commands can be inhibited during the SYSGEN process

    Called a PREPGEN

    Saved answer file can be corrected/modified for use later in live SYSGEN

New Features for Version 4.0

- Autoconfigure option determines correct hardware configuration

    Processor type

    CSR and vector addresses

- Standard feature Executive option produces a mapped system

    Most software options

    Supports all layered products

- Improved saved answer file support

    Saves responses to Phase II questions

    Supports named input and output saved answer files

    Can create a single saved answer file for Phase I and II

- A minimum of 28K words of memory is required for SYSGEN

- Directive commons increase POOL size

- Taskbuild .CMD and .ODL files are dynamically created

- A PLAS overlaid FCS resident library may be selected

- Either EDI or EDT may be selected for editing SYSGEN files

- Phase I can chain to Phase II without a break

PREPARING FOR SYSGEN

Distribution Kits

- There are several different kits available

    'Big Disk' distribution kit for RPØ4/Ø5/Ø6/Ø7, RMØ2/Ø3/Ø5, RM8Ø and RA8Ø

    RKØ6/Ø7 distribution kit

    RLØ1/Ø2 distribution kit

    RKØ5 distribution kit

- Distribution kit must be copied before SYSGEN using DSC

    Version 3.2 DSC cannot be used unless V 4.Ø patch has been applied (see release notes)

    Copy of standalone DSC supplied with kit can be used

Autopatch

- A collection of patches to the released system

    Issued at regular intervals

    Also may include patches to layered products

- Must always be applied to released version without previous patches

- Uses an indirect command file to generate patched system

Individual Patches

- Patching source files

    Used for Executive, driver and MCR patches

    Use Source Language Input Program (SLP)

    Create SLP command file
    Invoke SLP with command file

    For MACRO-11 sources, output for SLP must be:

    Assembled
    Task-built
    Integrated into system (Method depends on component patched)

    Patching object files

    Used for patching utilities

    Use Object Module Patch Utility (PAT)

    Create MACRO-11 source file for correction
    Assemble correction file
    Apply correction to old object file using PAT

    Output from PAT must be:

    Task-built
    Reinstalled into the system

- Patching task image files

    Infrequently used for operating system patches

    Frequently used for patching layered products

    Use Task/File Patch Program (ZAP)

    Create ZAP command file
    Invoke ZAP to apply patches to task image file

- Patches supplied by DIGITAL are cumulative

    Apply a patch only if you have already applied all previous patches

    Unless the patch states otherwise, apply patch to most recent version of the component

DETAILS OF SYSGEN

Phase 1

There are seven sections in Phase 1

Setup - How SYSGEN is to be run

Target Configuration - The environment for which the SYSGEN is being performed

Host Configuration - The system on which SYSGEN is being run

Executive Options - What features are desired in the new system

Terminal Driver - Which type of terminal driver and which features are desired

System Options - Which file processor, CLI support, and POOL monitoring support is desired

Peripheral Options - Specification of devices, controllers and their vectors, and CSRs

After questions have been answered, the Executive and drivers are assembled

Phase 2

There are eight sections in Phase 2

Setup - How SYSGEN Phase 2 is to be run

Library Commons Creation - Modifies SYSLIB, creates other libraries used in taskbuilding system tasks

Executive Taskbuild - Creates system images RSX11M.TSK, EXCOM1.TSK and EXCOM2.TSK

System Image Creation - Creates RSX11M.SYS from RSX11M.TSK

Full-Duplex Terminal Driver Taskbuild - Creates the full-duplex driver if selected

Loadable Driver Taskbuild - Creates all drivers not built into the Executive

Privileged Task Taskbuild - Builds privileged system tasks

System VMR - Modifies RSX11M.SYS, creating partitions, installing tasks, etc.

Phase 3

- Optional – May be performed when and as often as desired

- Used to build optional tasks not included in distribution kit

- Used to rebuild supplied tasks to:

    Change defaults

    Incorporate support for ANSI magtape and/or FCS big-buffering

    Use a resident FCS library

- Used to rebuild tasks after patching

- VMR must be used to remove and install new versions of tasks installed in the system image

EXECUTIVE OPTIONS

Standard Feature Executive

● Allows the generation of a system supporting all standard features

● Requires a mapped system

● Provides the following:

        DBMS-11 support
        FILES-11 ACP support
        RMS support
        Nonresident task support
        Loadable task loader
        Directive commons
        Memory management directives
        All other Executive directives
        Address checking support
        I/O rundown support
        Multiuser support
        ANSI Magtape support
        Loadable driver support
        AST support
        20K word Executive
        Error logging
        Powerfail recovery
        User written driver support
        System controlled partitions
        Round-robin scheduler and disk swapping
        Software write lock support
        Queue manager
        Shuffler
        Crash dump support
        Full-duplex terminal driver
        DCL and two user written CLIs
        POOL monitoring
        PLAS resident FCS
        'MIDDLE' F11ACP

The Terminal Driver

- Two possible terminal drivers:

    Half-duplex driver
    Full-duplex driver

- Half-duplex driver comes in three forms:

    Baseline
    Tailored
    Tailorable

- The baseline half-duplex driver

    Minimum terminal driver
    Not available with multiuser systems

- The tailored half-duplex driver features include:

    Automatic carriage/line feed
    Write with CTRL/O cancellation
    Breakthrough write
    CTRL/R rewrite
    Get multiple characteristics
    Get terminal driver options
    Read after prompt
    Read with no echo
    Read with special terminator
    CRT rubout support
    User terminal input buffering

- The tailorable driver allows selection of all possible terminal options

- The full-duplex driver includes:

    Task checkpoint during input
    Passes form feeds directly to the terminal
    Supports unsolicited input character AST (requires Executive AST support)
    Hold screen mode
    LA30P support
    Device independent cursor positioning
    Plus features included in the tailored half-duplex driver

- Full-duplex driver requires a mapped system with loadable driver support

117

Choosing a File Processor

- Several different versions of F11ACP available

- Versions differ in amount of overlaying and in number of internal buffers

- See discussion in module 'I/O Processing' for descriptions of available ACPS

USING VMR

Overview of VMR

- Can perform many of the same tasks as MCR

    Define partitions

    Load drivers

    Install tasks

- Commands operate on system image file

- Used to perform system initialization for needed features

    Set up partitions

    Load standard drivers

    Install system tasks and utilities

On a Mapped System

- SYSGEN generates a VMR command file which should be edited to produce the system desired

- All needed partitions are defined

   Driver partition

   Directive commons

   All leftover space goes in GEN

- Commands for partitions needed by application tasks must be added to the command file

- All DEC-supplied drivers are loaded

- All important system tasks are installed

GEN	
	321400
FCPPAR	
	275400
FCSRES	
	242700
SYSPAR	
	232600
DRVPAR	
	206200
TTPAR	
	146200
LDRPAR	
	143400
EXCOM2	
	134500
EXCOM1	
	120000
EXECUTIVE	

Figure 5-1 Partition Structure on Mapped Systems

On an Unmapped System

- Modification of SYSVMR.CMD recommended on unmapped system

  It is not possible for SYSVMR.CMD to create an optimal partition structure

- For unmapped 16K to 24K systems, VMR:

  Creates partition GEN and installs all tasks into it

  Length is always 40000 bytes

  Creates partition SYSPAR

  Length is 1000 bytes if FCPMIN chosen
  Length is 1200 bytes if FCPSML chosen

  Creates partition SPLPAR if PRT chosen

  Length is 10400 bytes

- For unmapped 24K to 28K systems, VMR:

  Creates partition GEN and installs all tasks into it

  Length is always 40000 bytes

  Creates partition SYSPAR

  Length is 1000 bytes if FCPMIN chosen
  Length is 1200 bytes if FCPSML chosen

  Creates partition PAR14K

  Length is 70000 bytes

  Creates partition SPLPAR if PRT selected

  Length is 10400 bytes
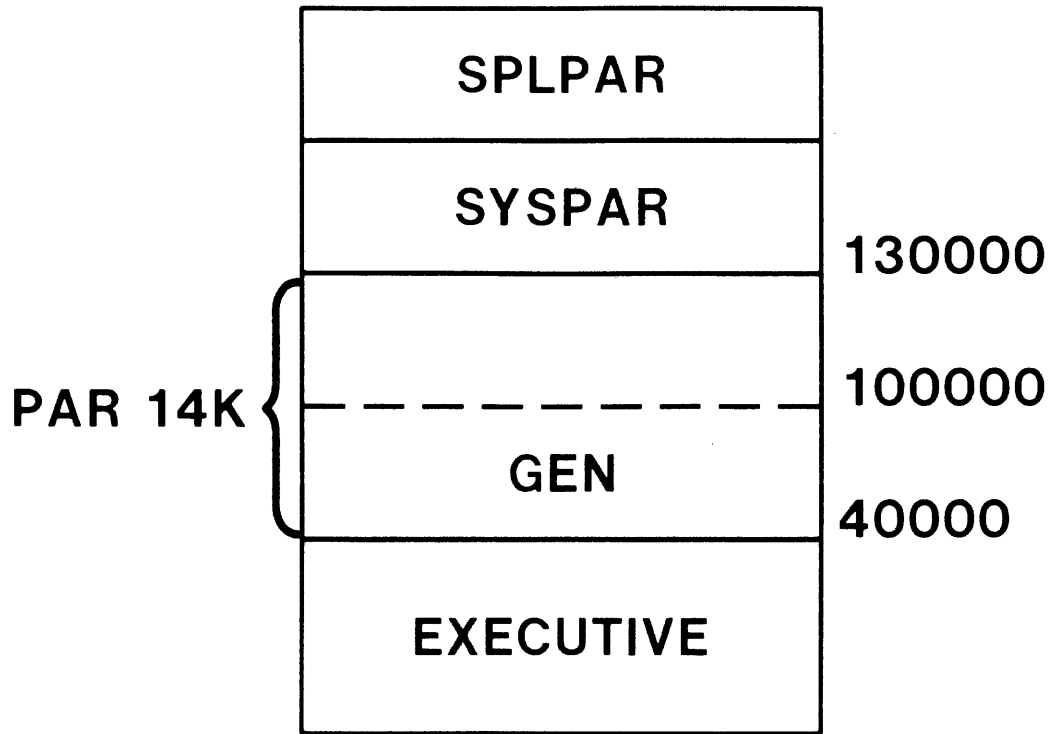
Figure 5-2 Partition Structure on Unmapped 24K to 28K Systems

SPLPAR	
SYSPAR	
	100000
GEN	
	40000
EXECUTIVE	
	0
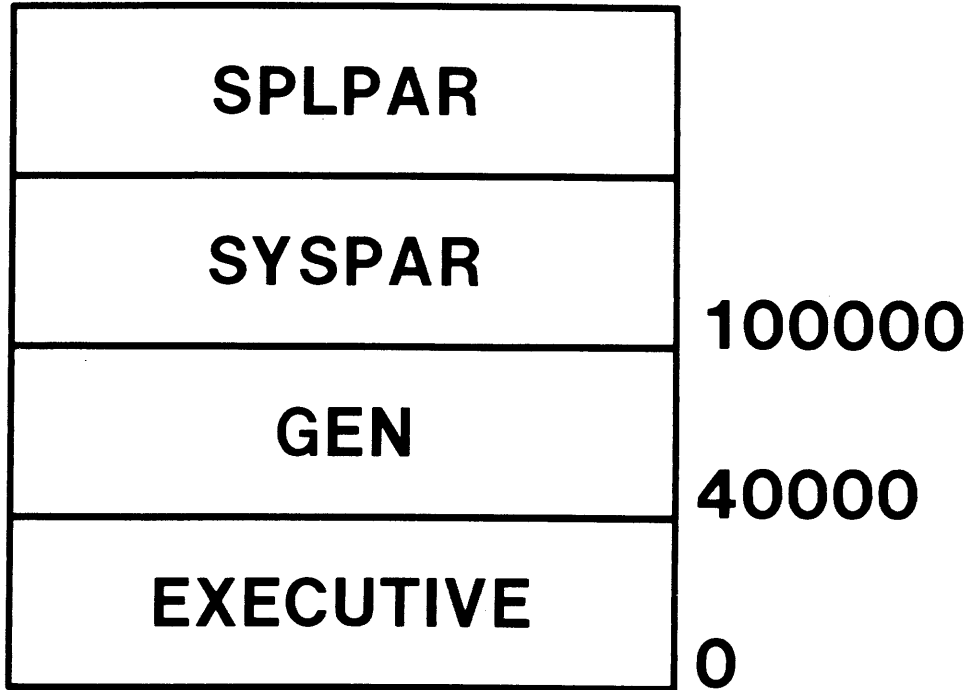
Figure 5-3   Partition Structure on Unmapped 16K to 24K Systems

SAVING THE NEW SYSTEM

- Copies the current system image into the system image file from which the system was booted

- Used to make a system hardware bootable

- System must be saved before compressed by DSC or BRU

- Requirements for saving a system

  SAV must run from CO:

  Error logging is not active

  All tasks are installed from LB:

  No checkpoint files are active

  No volumes are mounted except the load device

  The load device can be successfully dismounted

  No tasks have outstanding I/O

  No task is connected to interrupts

  All drivers, active tasks and fixed tasks reside within the saved area of memory

- Saving a system

  Lowest part of memory copied into buffer in SAV task

  Secondary bootstrap program loaded into low memory

  Special driver used to load system to and from disk

    Cannot use interrupts

    Same special drivers used by SAV and BOOT

Can create a bootstrap program within the volume bootblock

Reads in first block in system image file

Transfers control to secondary bootstrap in block just
loaded into memory

Before writing system image to disk, converts logical
block number in TCB into file ID

When system booted, file IDs in TCBs converted back into
logical block number
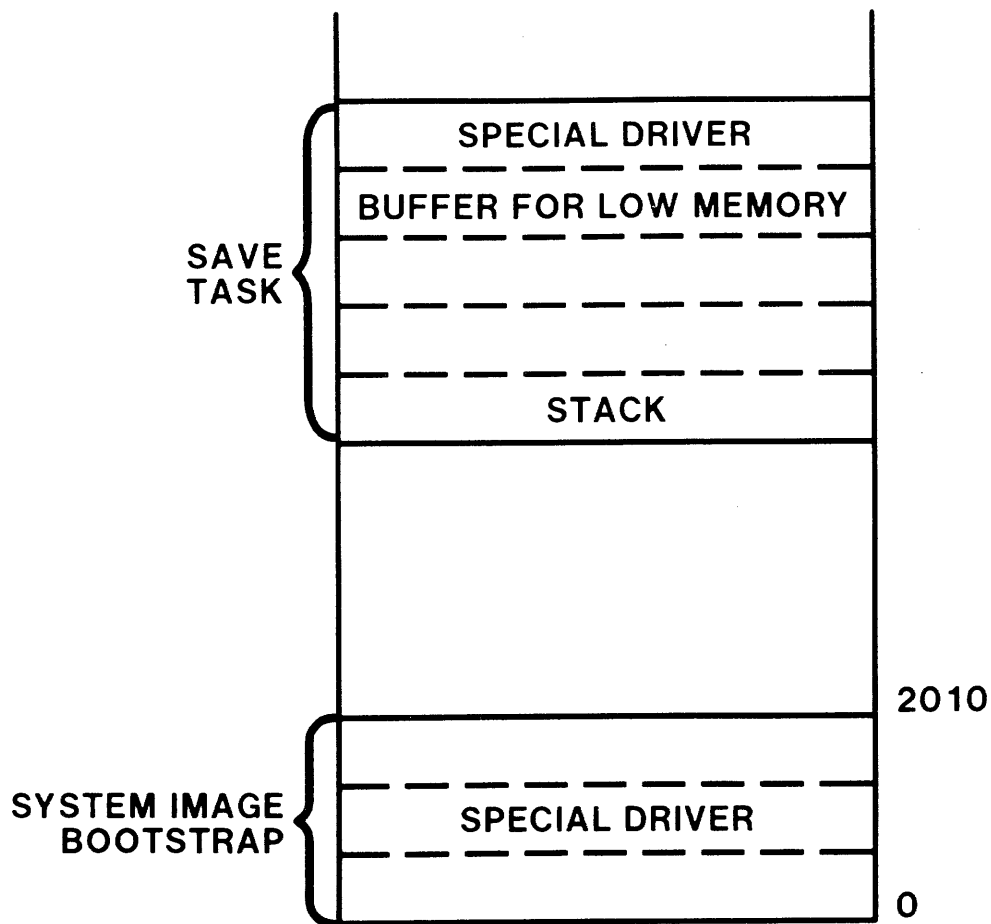
Figure 5-4   Saved System Image

PHYSICAL AND VIRTUAL MEMORY

INTRODUCTION

The most important resource managed by an operating system is physical memory. To manage memory, an operating system must define subdivisions of memory, allocate space to tasks, and optimize memory use. This module examines the data structures used to perform these tasks.

Since the PDP-11 uses 16-bit words, only 32K words of physical memory can be addressed unless some form of address relocation is provided. On the PDP-11, the KT-11 memory management hardware provides the mechanism for relocation. This module also explains the hardware and processing used.

OBJECTIVES

1. Diagram the data structures used by the Executive to control physical memory.

2. List the steps in checkpointing a task.

3. Describe the memory management hardware.

CONCEPTS

Physical Addresses

- A contiguous series of word addressable hardware locations

- Defines main memory and peripheral device registers

Virtual Addresses

- Set of addresses used within a task

- Program counter is a 16-bit register

    User task can reference addresses between 0 and 177777 octal

    Virtual addresses are local to a task

Unmapped Systems

- 32K words of addressing space

- Maximum physical memory is 28K words

- Programs built for specific memory location

- No memory protection
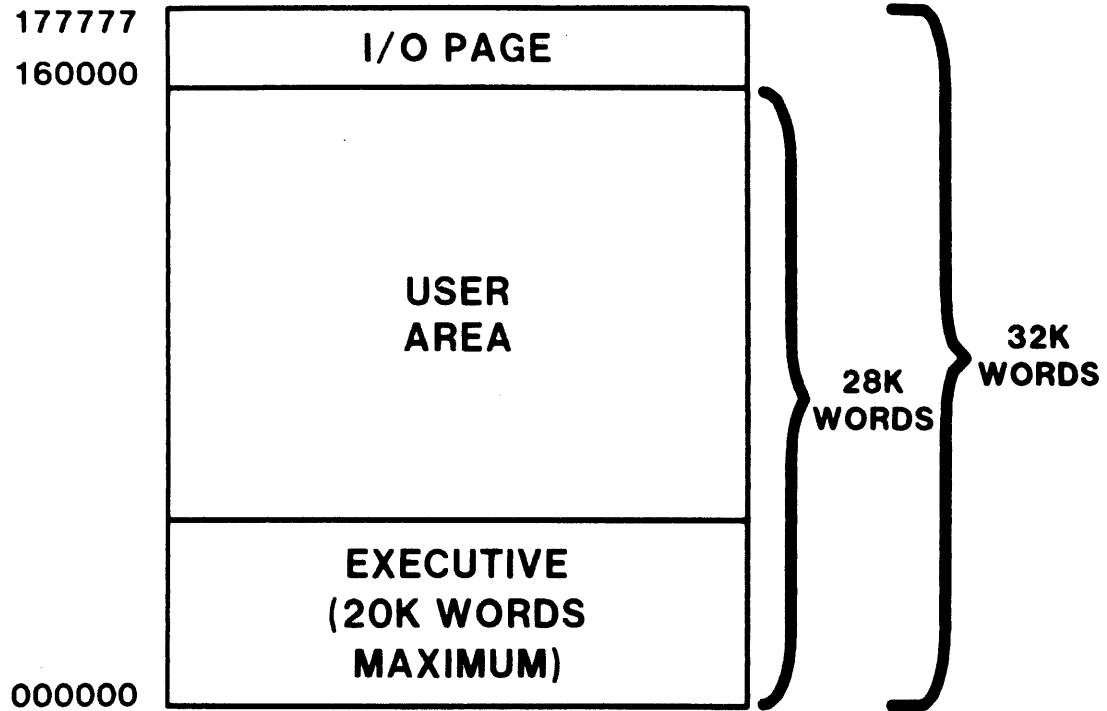
Figure 6-1   Typical Unmapped System

Mapped Systems

- KT-11 hardware is present

- Two types of mapped systems

      18-bit addresses
          128K words of addressing space
          124K words of physical memory

      22-bit addresses
          2048K words of addressing space
          1920K words of physical memory

- 16-bit virtual address converted to 18- or 22-bit physical addresses

- Virtual addresses range from 0 to 32767

- Relocation and memory protection provided
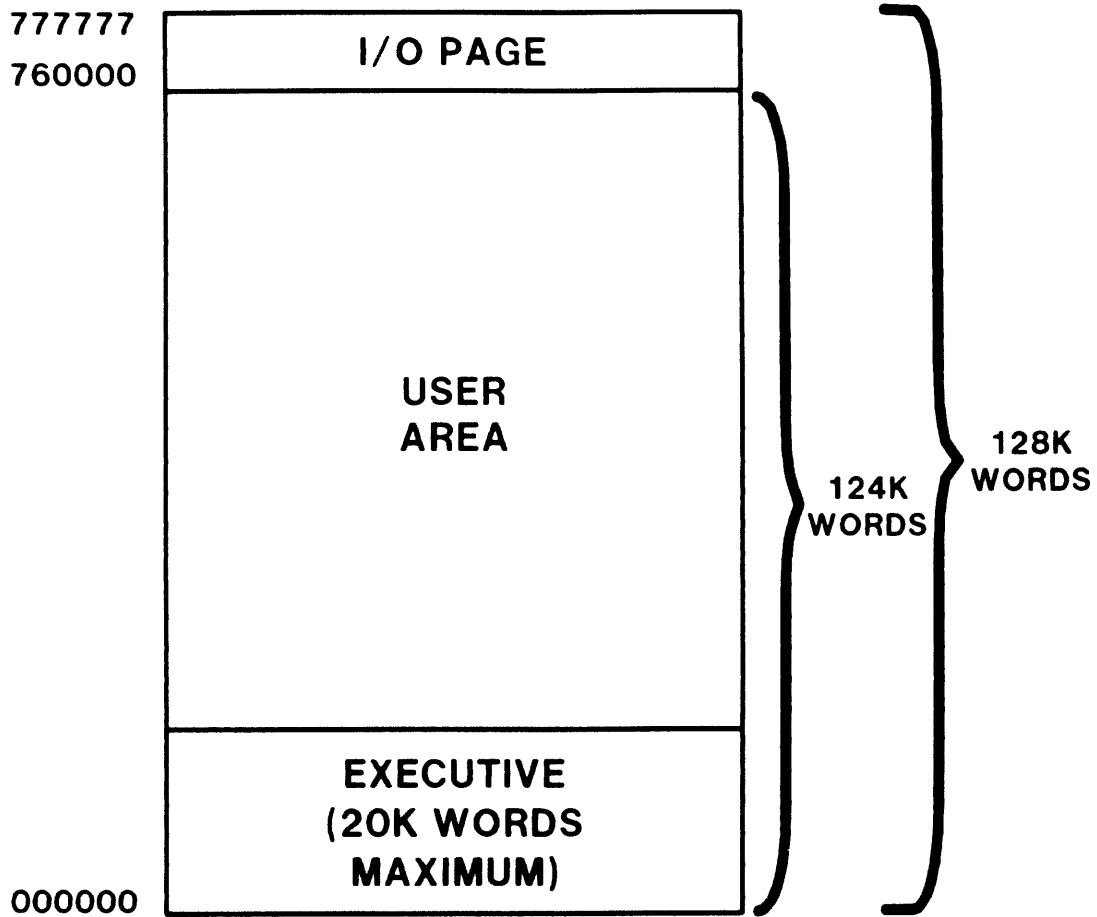
Figure 6-2 Typical Mapped System with 18-bit Addressing
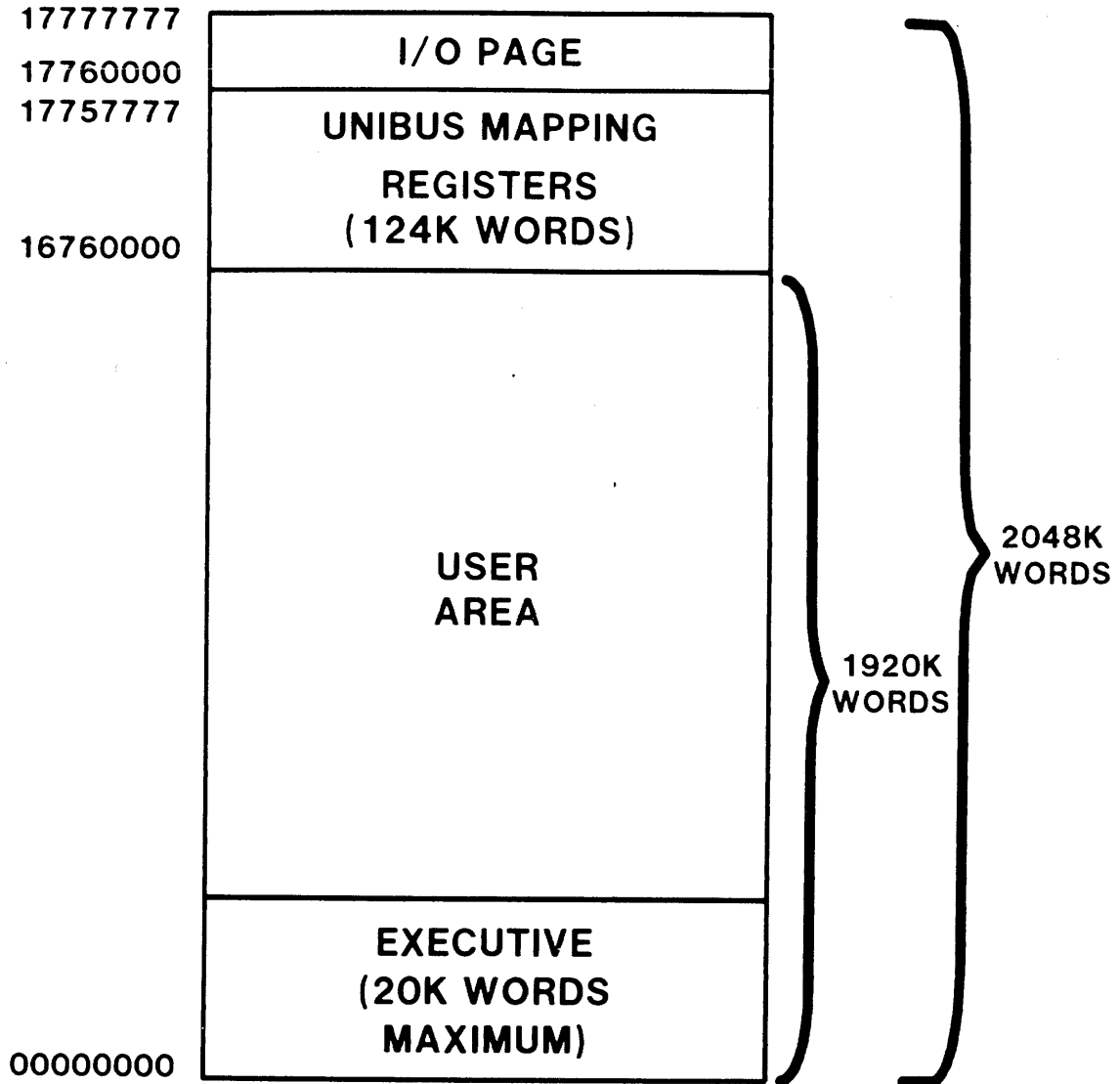
Figure 6-3   Typical Mapped System with 22-bit Addressing

MEMORY MANAGEMENT

Overview

● Basic functions

  Perform memory relocation/mapping

  Provide extended memory addressing capability

● Task is mapped in pages

  Page length is from 1 to 128 memory blocks

  Memory block is 32 words in length (100 octal bytes)

Memory Management Hardware

- Several sets of relocation registers: Active Page Registers (APRs)

- Each APR consists of two 16-bit words

    Page Address Register (PAR)
        Contains a physical memory address in units of 32 words memory blocks
        Used to relocate 4K words of virtual memory (one page)

    Page Descriptor Register (PDR)
        Contains the size of the virtual memory block to be relocated
        Provides control of access rights

- Every mapped system contains at least two sets of eight APRs
        KERNEL Mode APRs
        USER Mode APRs

- Larger processors have more APRs
        APRs for KERNEL and USER mode Instruction and Data space
        Supervisor mode Instruction and Data space APRs

- The Processor Status Word (PS) determines the APRs in use

- Four memory management registers (MMR0, MMR1, MMR2, and MMR3) control processor use of memory management

    MMR0 - Error flags
        Page number whose reference caused error
        Other status flags

    MMR1 - Only on 11/44 and 11/70
        Records any autoincrement/decrement of general purpose registers

    MMR2 - The Virtual Address Program Counter

    MMR3 - Only on 11/44 and 11/70
        Enables/disables
            Use of D space APRs
            22-bit mapping
            UNIBUS mapping
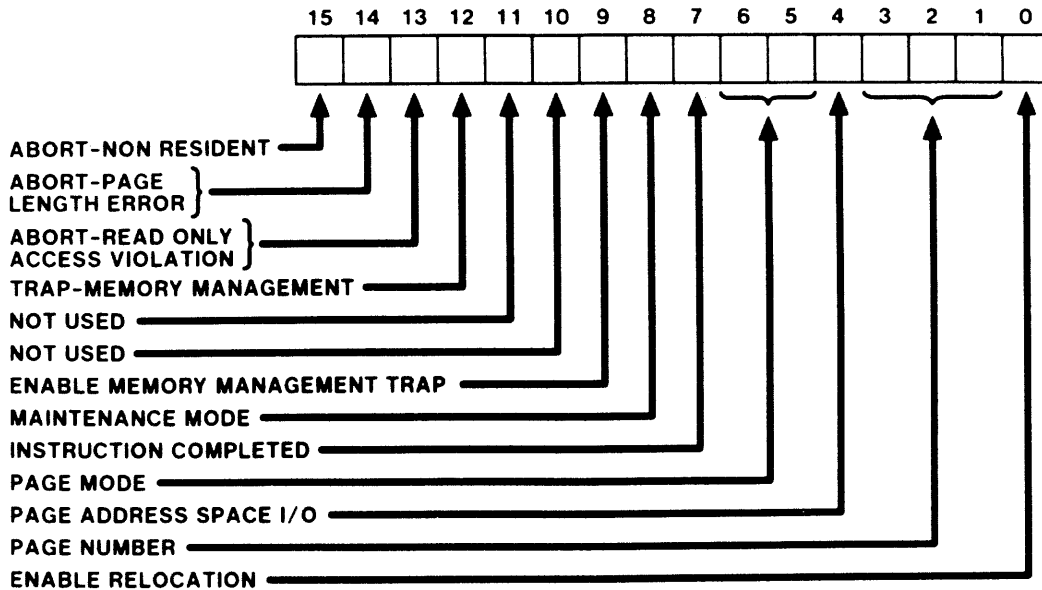
ABORT-NON RESIDENT
ABORT-PAGE LENGTH ERROR
ABORT-READ ONLY ACCESS VIOLATION
TRAP-MEMORY MANAGEMENT
NOT USED
NOT USED
ENABLE MEMORY MANAGEMENT TRAP
MAINTENANCE MODE
INSTRUCTION COMPLETED
PAGE MODE
PAGE ADDRESS SPACE I/O
PAGE NUMBER
ENABLE RELOCATION

MEMORY MANAGEMENT REGISTER #0 (MMRO)

AMOUNT CHANGED (2'S COMPLEMENT) | REGISTER NUMBER | AMOUNT CHANGED (2'S COMPLEMENT) | REGISTER NUMBER

MEMORY MANAGEMENT REGISTER #1 (MMR1)

ENABLE UNIBUS MAP
ENABLE 22-BIT MAPPING
KERNEL
SUPERVISOR
USER
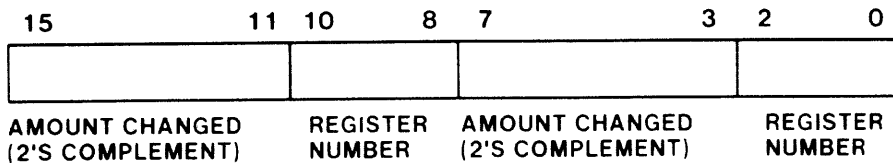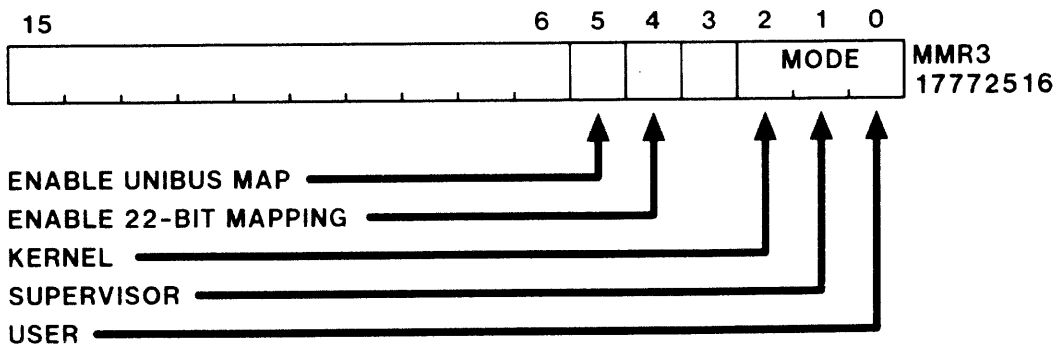
MEMORY MANAGEMENT REGISTER #3 (MMR3)

Figure 6-4   Registers Used by Memory Management

138

THE PAGE ADDRESS REGISTER

THE PAGE DESCRIPTOR REGISTER
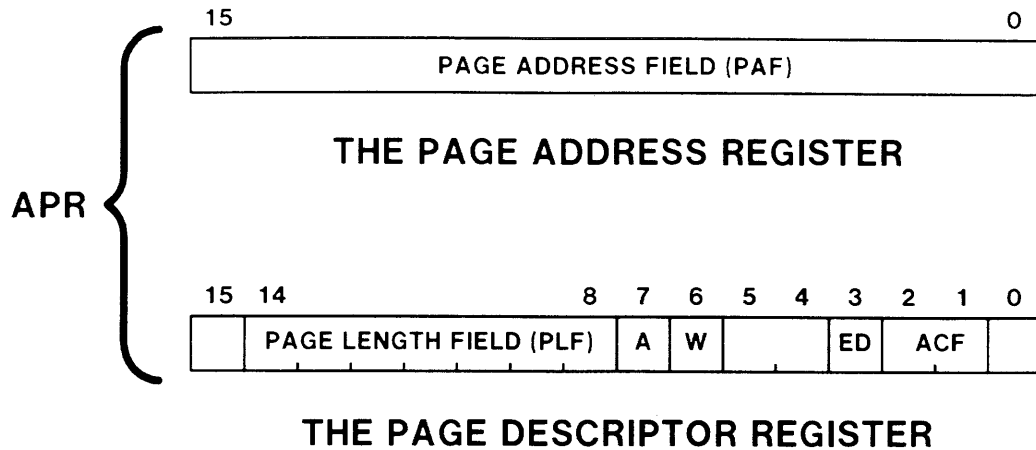
ACF - ACCESS CONTROL FIELD

ED - EXPANSION DIRECTION

A
W **} - ACCESS INFORMATION BITS**

PLF - PAGE LENGTH FIELD

Figure 6-5 Active Page Registers

PHYSICAL MEMORY

PAR NUMBER	VIRTUAL ADDRESSES	PARs		
			4K WORDS	473400
			4K WORDS	453400
7	160000-177777	002300		
6	140000-157777	002100		
5	120000-137777	000000	4K WORDS	324200
4	100000-117777	000166		
3	060000-077777	000000		
2	040000-057777	003242	4K WORDS	230000
1	020000-037777	004734	4K WORDS	210000
0	000000-017777	004534		
			4K WORDS	16600
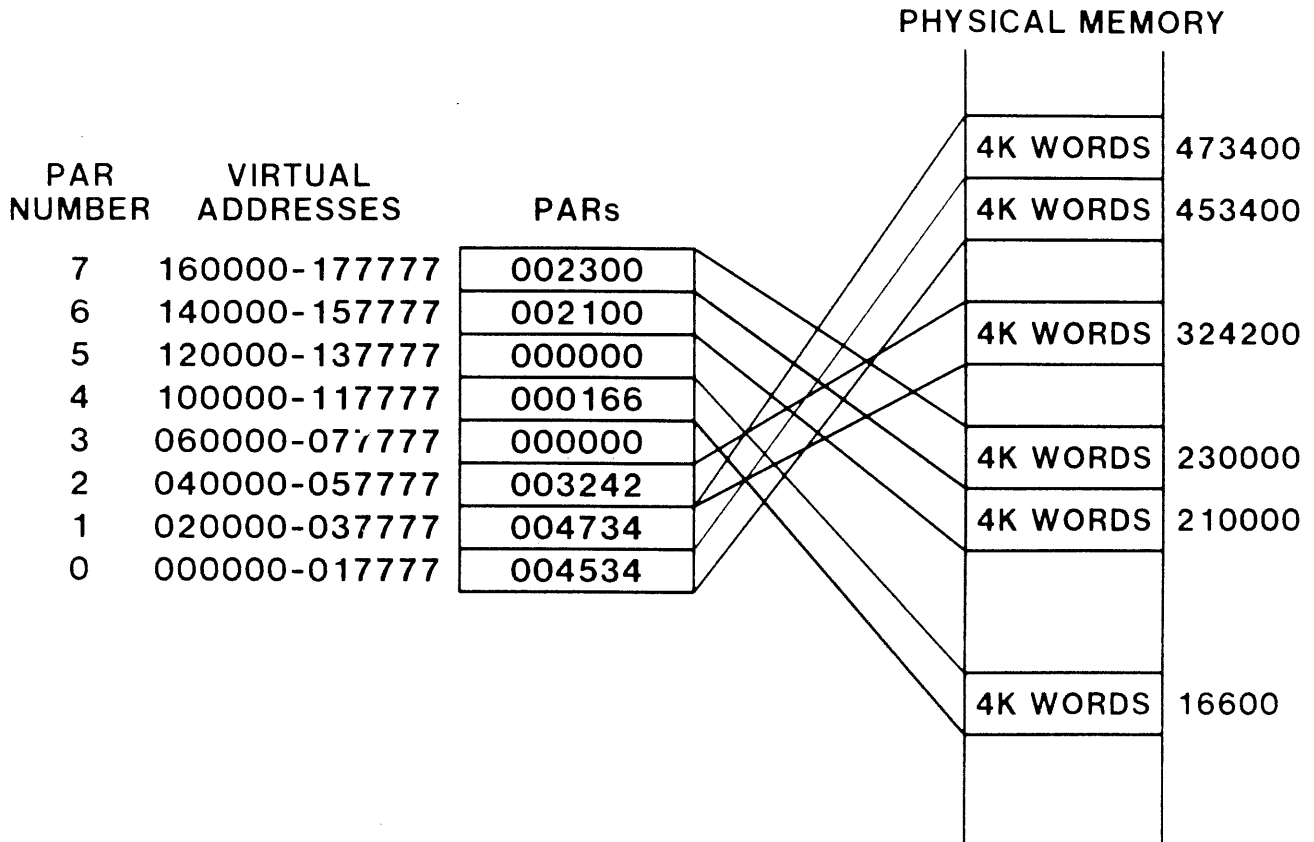
Figure 6-6   Page Address Registers

Software Use of Memory Management

- RSX-11M uses KERNEL and USER instruction space APRs

- KERNEL APRs are loaded so that

  APRs 0 through 3 or 4 point to Executive code and data

      4 APRs for 16K word Executives
      5 APRs for 20K word Executives

  APR 7 points to the I/O page

  Other APRs are dynamically mapped as needed

- User tasks are mapped using USER APRs

  If task does not map to the Executive

      APRs are used consecutively starting at APR 0

  If task maps to the Executive

      APRs 0 through 3 or 4 are the same as the KERNEL

      APRs 0 through 3 or 4

      APR 7 points to the I/O page

      Other USER APRs are used to map the user task
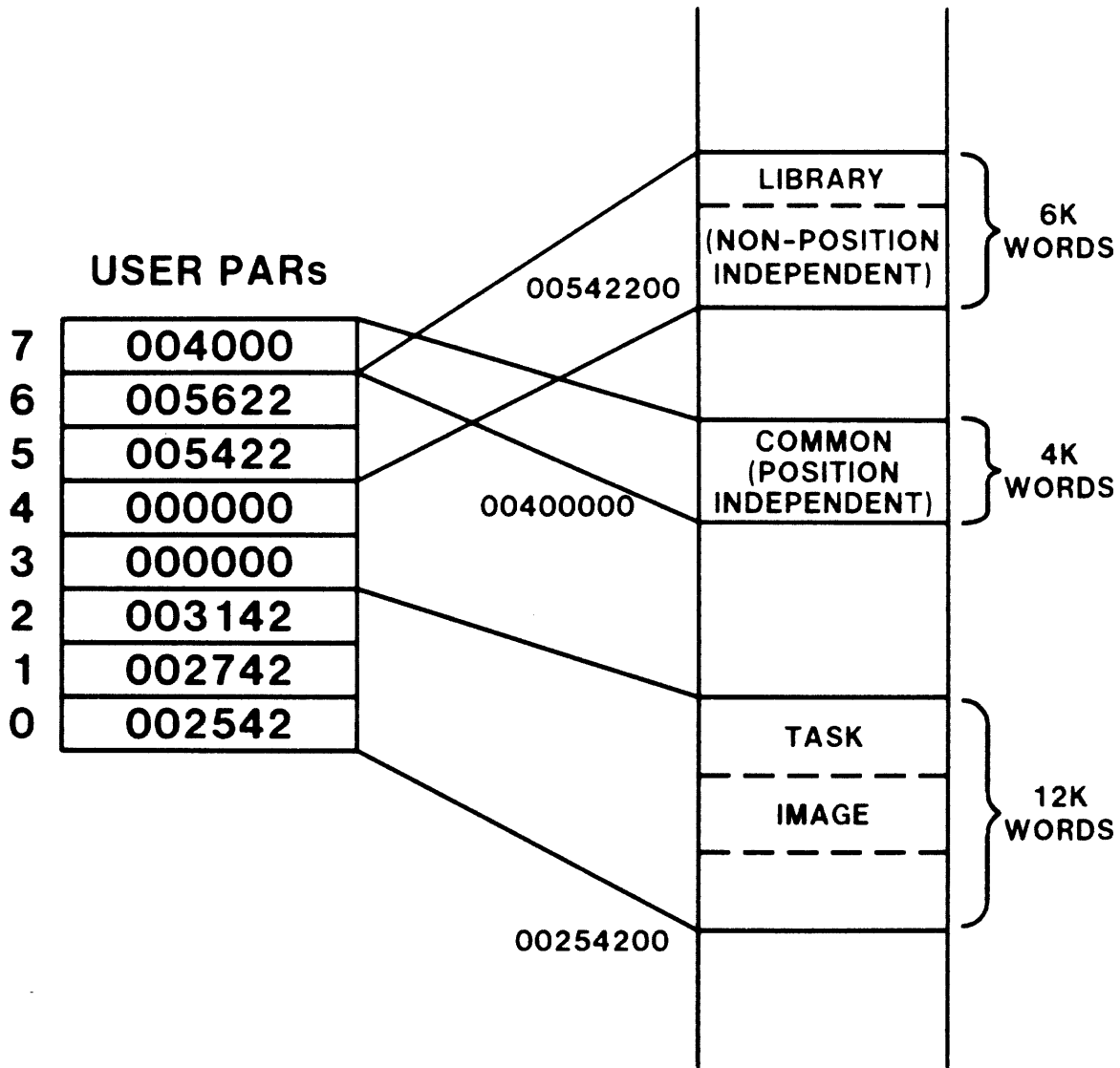
Figure 6-7   Use of Memory Management by a Nonprivileged Task

001600 - 16-BIT
007600 - 18-BIT
177600 - 22-BIT

I/O PAGE

4K
WORDS

KERNEL PARs

7	
6	000000
5	000000
4	
3	000600
2	000400
1	000200
0	0000

4K WORDS

16K WORD
EXECUTIVE

20K
WORDS

000000 - 16K EXECUTIVE
001000 - 20K EXECUTIVE
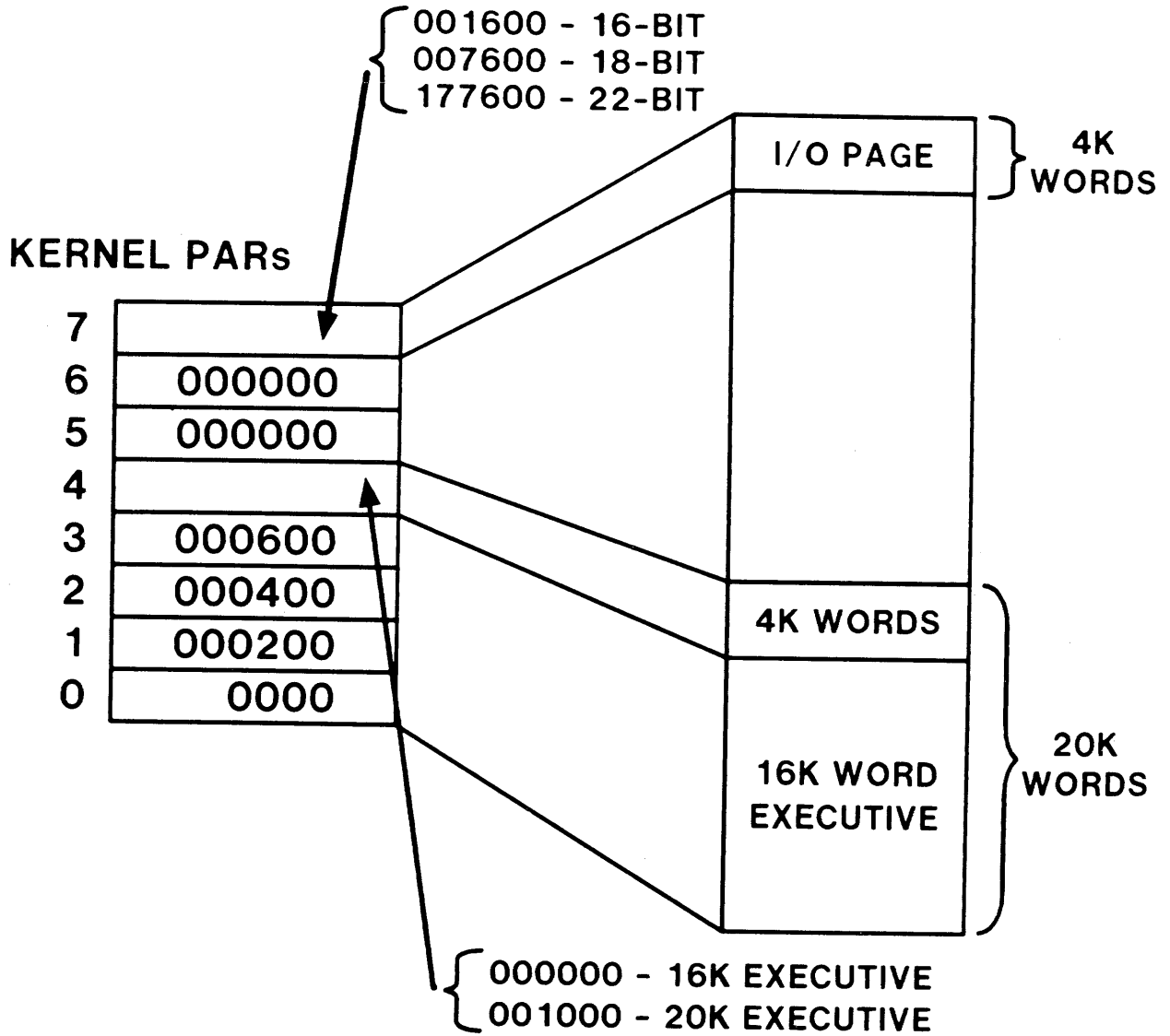
Figure 6-8 Use of Memory Management by the Executive
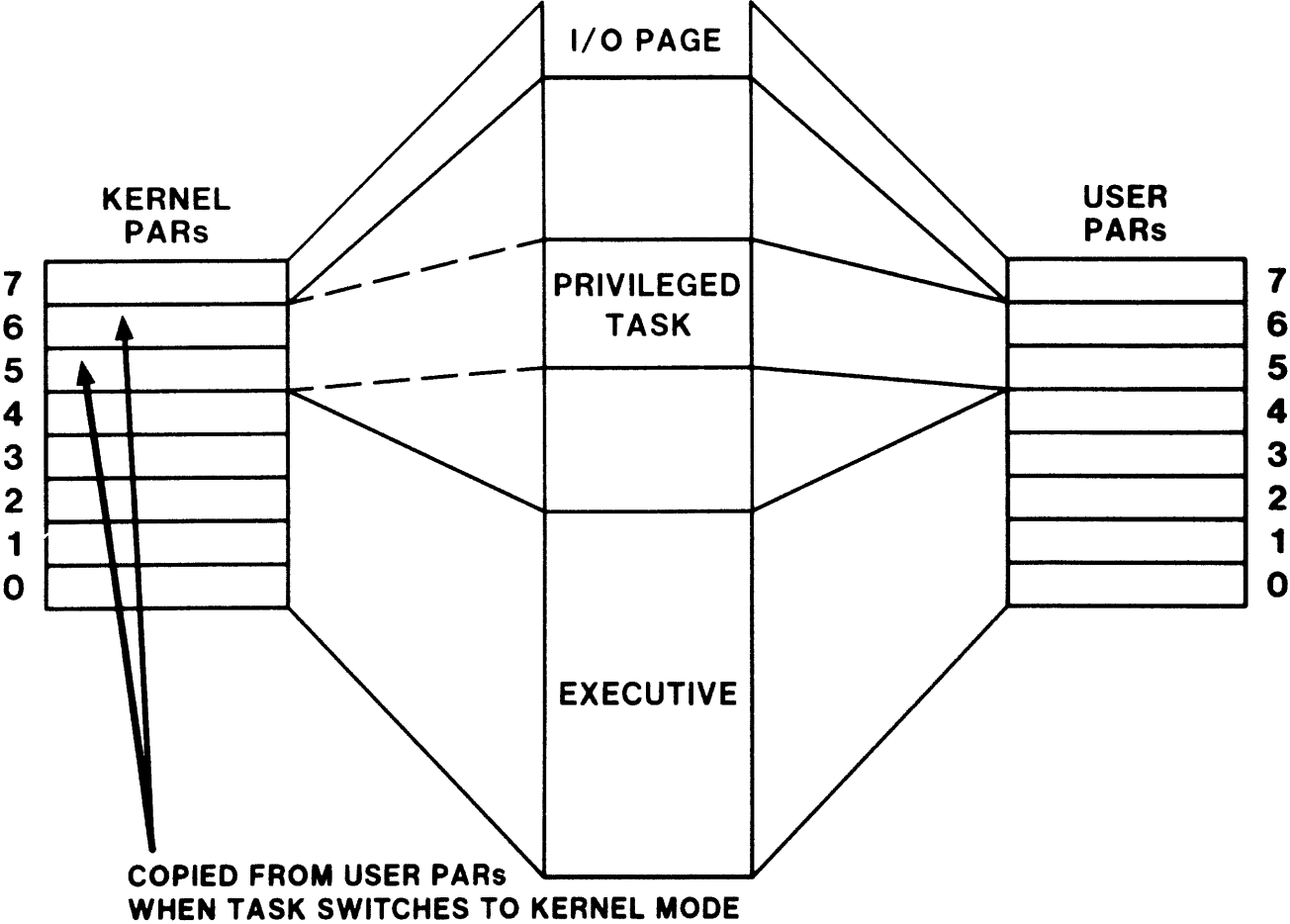
Figure 6-9   Use of Memory Management by a Privileged Task

ALLOCATING PHYSICAL MEMORY

Partitions

Task Partitions

- Controlled by user

- May be divided into subpartitions

- May be built to run in main partition or in one of the subpartitions

- Controlled by busy word P.BUSY in the PCB of the main partiton

Common Partitions

- Used for shared data or code

- Linked to task statically by taskbuilder or dynamically by using memory management directives

Device Partitions

- Pseudo-areas whose addresses overlay the I/O page

- Allow nonprivileged tasks to access the I/O page

System-Controlled Partitions

- Used to hold executable tasks, device drivers, and dynamic regions

- Space allocated as needed by the Executive

- Tasks loaded into dynamically created subpartitions

PCB OF
MAIN PARTITION

PCB OF 1ST
SUBPARTITION

PCB OF 2ND
SUBPARTITION

PCB OF 3RD
SUBPARTITION

P.SUB

P.SUB

P.MAIN

P.SUB

P.MAIN

P.SUB = 0

P.MAIN

| 01010000 | 10000000 | P.BUSY |

| 01000000 | 01000000 | P.BUSY |

| 00000000 | 00100000 | P.BUSY |

| 00010000 | 00010000 | P.BUSY |

MAIN PARTITION
NOT BUSY

1ST SUBPARTITION
BUSY

2ND SUBPARTITION
NOT BUSY

3RD SUBPARTITION
BUSY
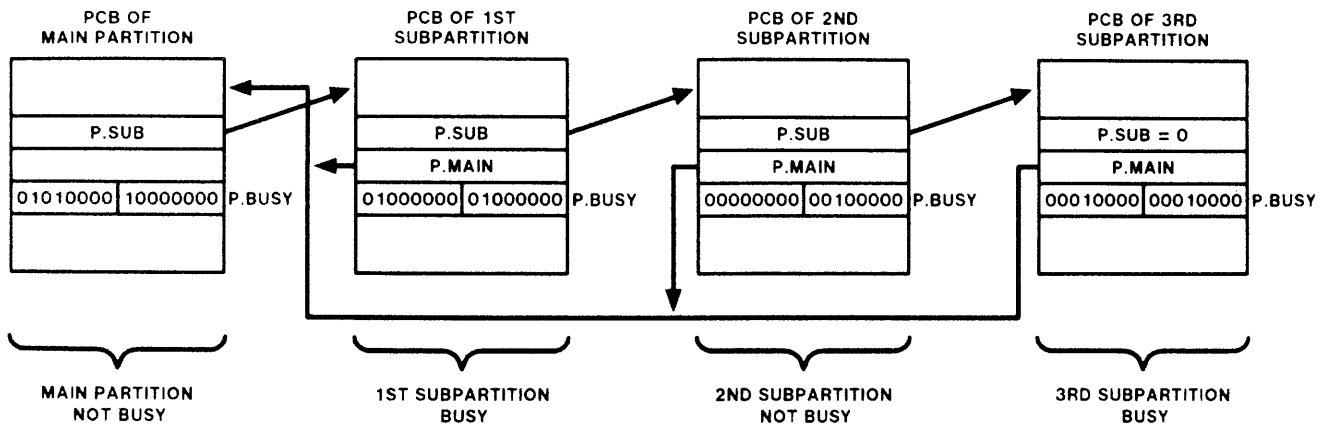
Figure 6-10   Subpartitions of a Task Partition

The Partition Wait Queue

- Linked list of TCBs of tasks waiting for space in partition

- Ordered by descending task priority

- When TCB is placed in wait queue of system controlled partition, subpartition PCB is created but not linked to partition list

- First task in queue is allocated space before others
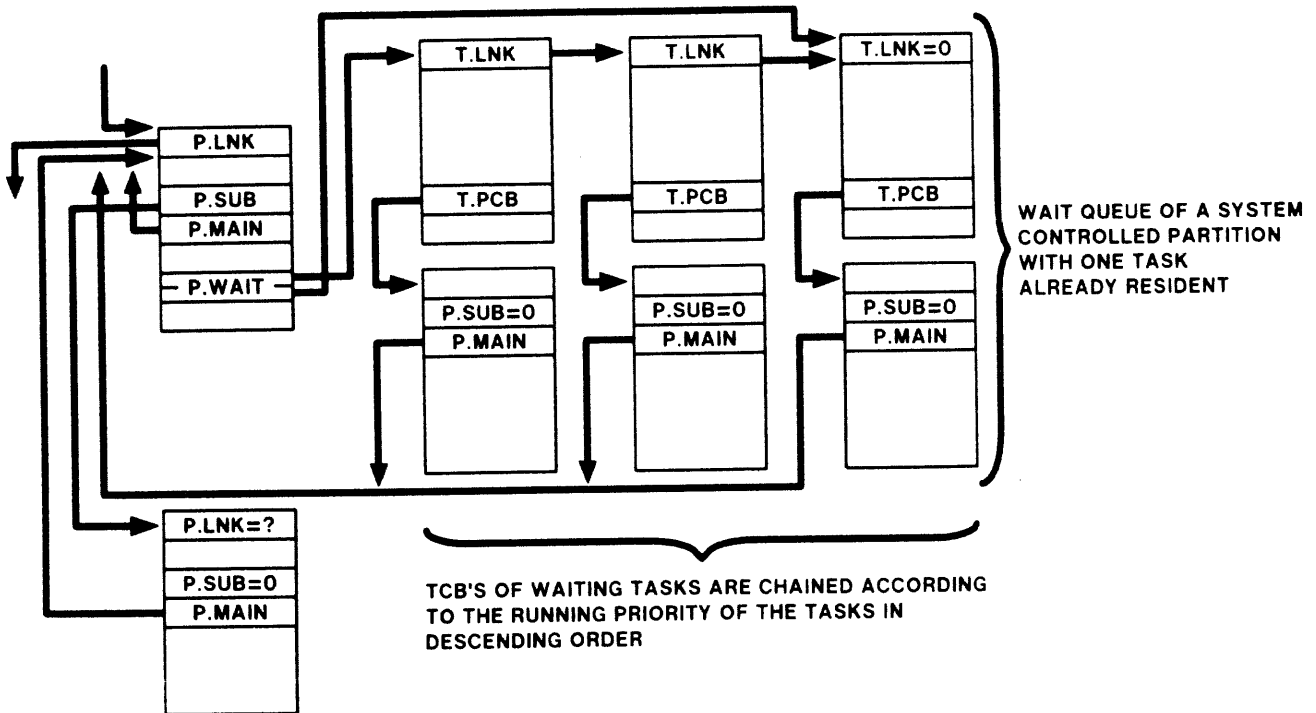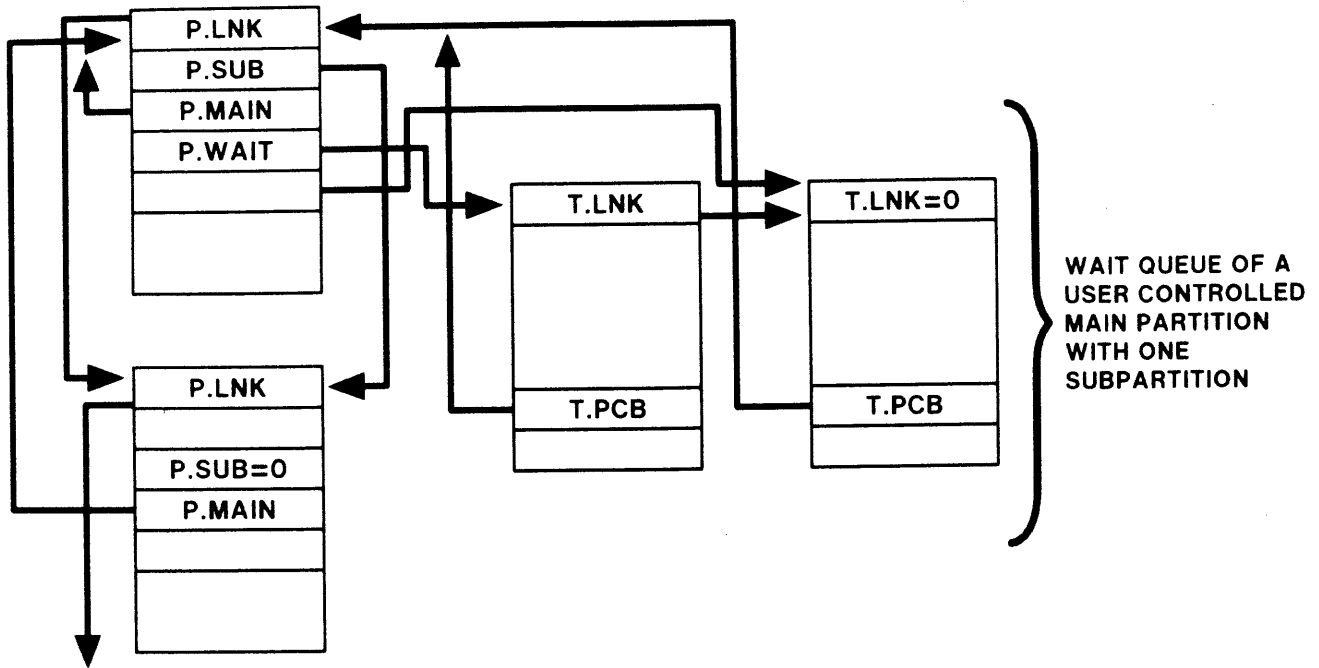
Figure 6-11   The Partition Wait Queue

USING PHYSICAL MEMORY

Loading Tasks

- Task TCB placed in loader queue after space allocated

- Loading performed by loader task, LOADR

- Loader task uses loader queue to decide which task to load next

    Utility link word, T.LNK, used to link TCBs in queue

    Location T.RCVL in the loader TCB is the queue listhead

- Steps in loading:

    Executive places TCB of task to be loaded in the loader queue (LOADRs receive queue)

    Executive unstops the loader

    Loader scans queue and loads task

    When queue is empty, loader calls $STPCT to stop itself

Checkpointing

● Task is copied from memory to space on a disk volume

Allows a higher priority task to execute

Copied either to a reserved area in task image file or to a system checkpoint file

Reserved area only used if system checkpoint files are full

● Steps in checkpointing a task

Test to see if task should be checkpointed

Initiate the checkpoint operation

Allocate checkpoint space and write the task to disk
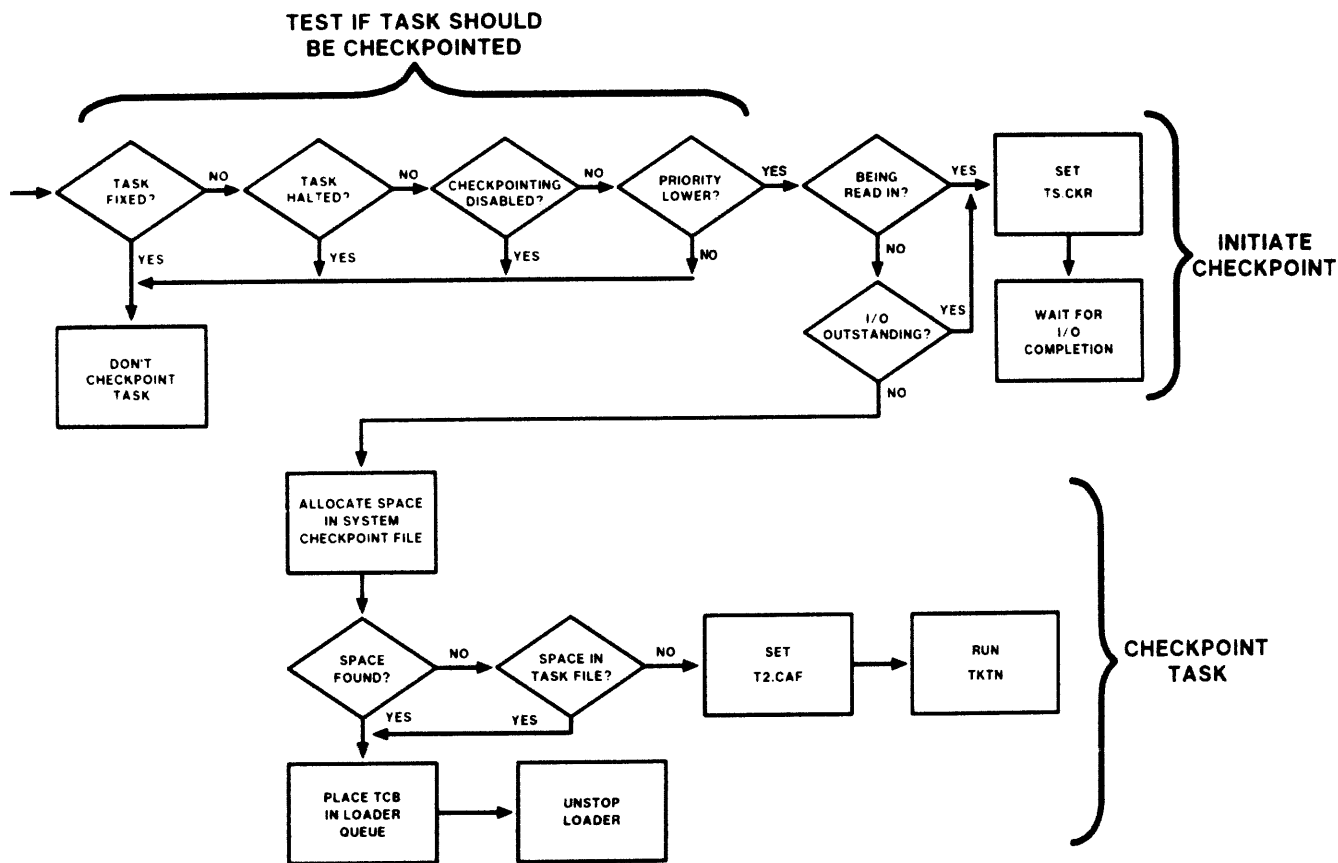
Figure 6-12 Steps in Checkpointing a Task

POINTER TO NEXT CHECKPOINT PCB	0 P.LNK
UCB ADDRESS OF CHECKPOINT DEVICE	2 P.PRI
——— LBN OF CHECKPOINT FILE ———	4 P.NAM 6
POINTER TO NEXT CHECKPOINT SUBPARTITION CB	10 P.SUB
POINTER TO CHECKPOINT MAIN PARTITION CB	12 P.MAIN
RELATIVE BLOCK NO. WITHIN CHECKPOINT FILE	14 P.REL
SIZE OF CHECKPOINT TASK (IN DISK BLOCKS)	16 P.SIZE

CHECKPOINT PARTITION CONTROL BLOCK

Figure 6-13 System Checkpoint Files: The Checkpoint PCB
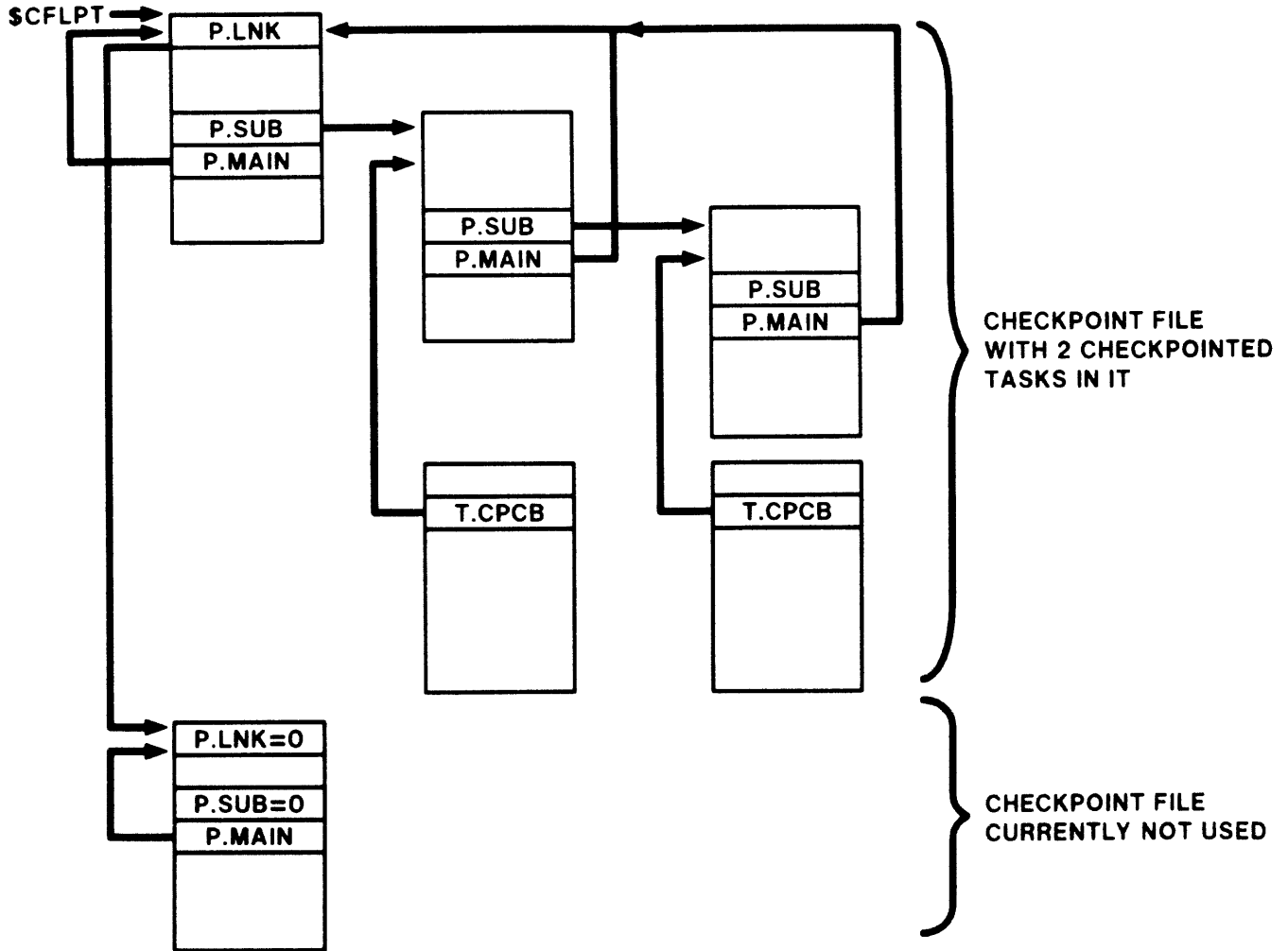
Figure 6-13 System Checkpoint Files: PCB Linkages

Shuffling

- Consolidates holes in a system-controlled partition

- Shuffling performed by shuffler task - SHUFL

- Performed in two passes over a system-controlled partition

Shuffler Pass One

- Partition list searched, from $PARHD, for system-controlled partition

- If task is in wait queue of system-controlled partition, shuffle operation is initiated

- Shuffle operation:

    Find next hole (first?)

    Find region following hole

    Determine if region is shufflable

    If shufflable, checkpointable and stopped, or blocked by MCR BLOCK command, task is checkpointed

    If shufflable, move region and contents to bottom of hole

    When all regions have been shuffled, invoke $NXTSK to reallocate space

- Nonshufflable regions

    Loaded device drivers

    Tasks connected to interrupt vectors

    Tasks fixed for parity errors

    Tasks with long outstanding I/O (did not complete in 1/2 second)

    Dynamic regions

    If nonshufflable region is found, next hole is found and process continues

- If pass one completes and the partition wait queue is not empty, pass two is invoked.

Shuffler Pass Two

- Attempts to gain space by finding a contiguous sequence of holes and lower priority checkpointable tasks

- Checkpoint operations initiated by calls to $ICHKP

- When checkpointing is complete, pass one is rerun
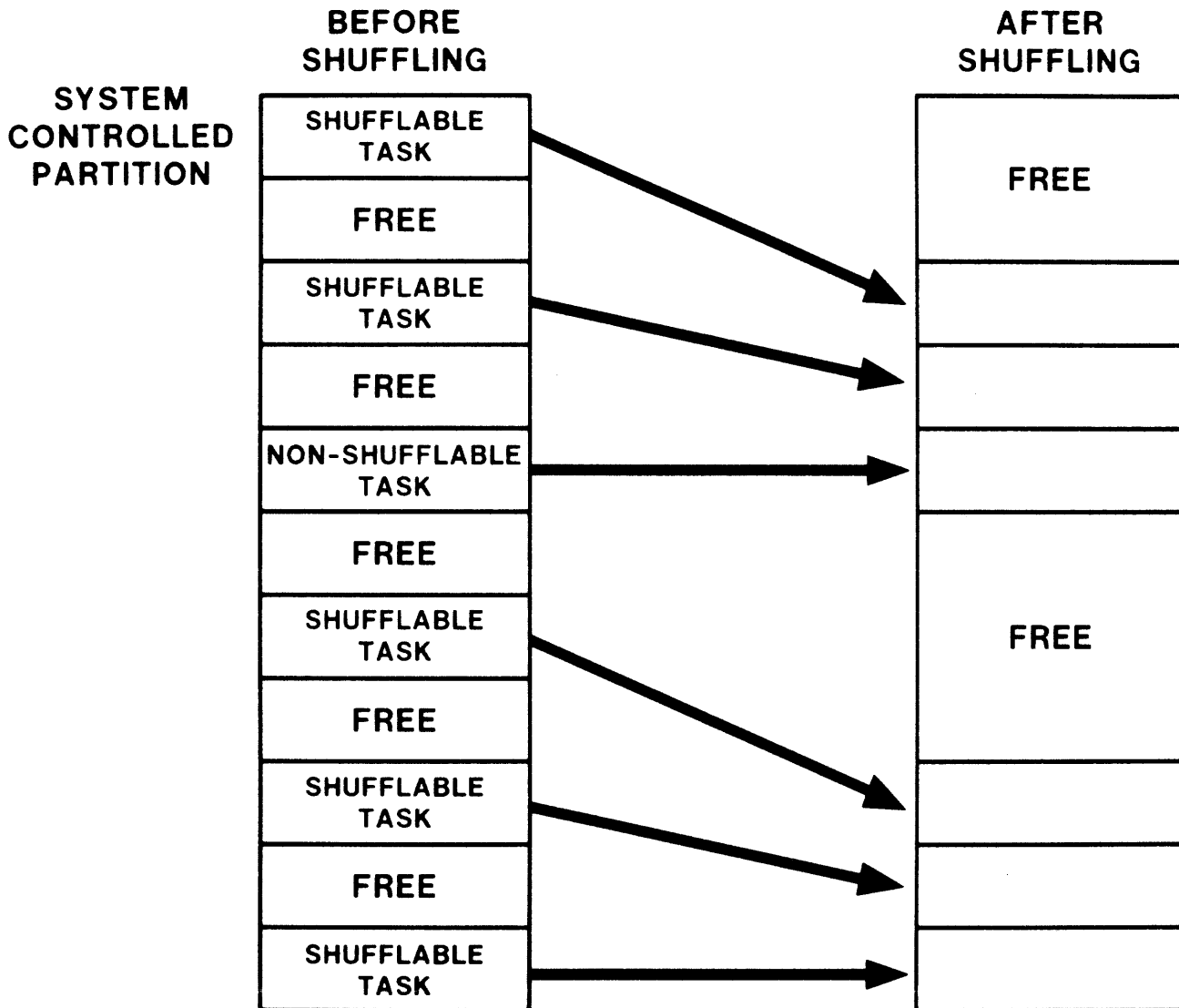
Figure 6-14   Shuffling Tasks

USING VIRTUAL MEMORY

Attaching

- Makes region available to attaching task

- Steps in attaching to a region:

    An Attachment Descriptor Block (ADB) is created

    ADB is linked to the PCB for the region using A.PCB in the ADB, and P.ATT in the PCB

    ADB is linked to the TCB for the task using A.TCB in the ADB, and T.ATT in the TCB

    ADB is linked into the ADB list for the partition using A.PCBL in the ADB

    ADB is linked into the ADB list for the task using A.TCBL in the ADB

PCB ATTACHMENT QUEUE THREAD WORD		0 A.PCBL
A.IOC 1 I/O COUNT THROUGH THIS DESCRIPTOR	PRIORITY OF ATTACHED TASK	2 A.PRI
TCB ADDRESS OF ATTACHED TASK		4 A.TCB
TCB ATTACHMENT QUEUE THREAD WORD		6 A.TCBL
A.MPCT 7 MAPPING COUNT OF TASK THROUGH THIS DESCRIPTOR	STATUS BYTE	10 A.STAT
PCB ADDRESS OF ATTACHED TASK		12 A.PCB
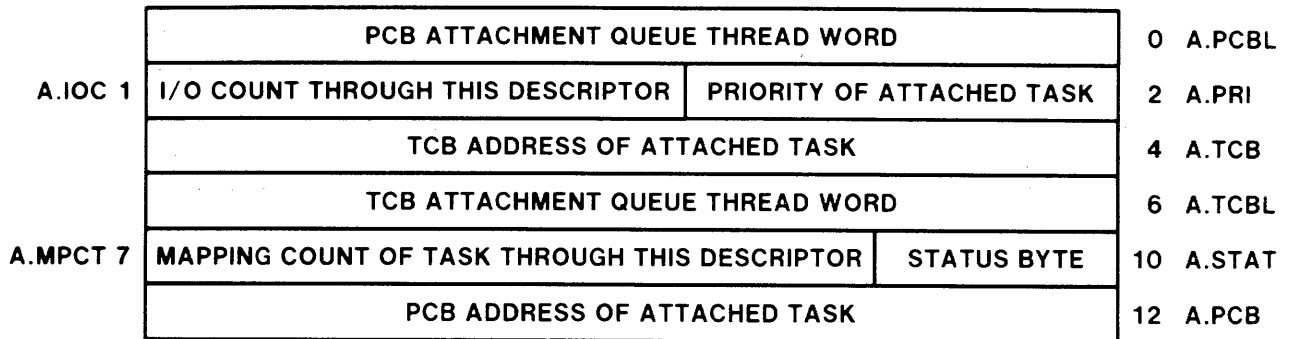
ATTACHMENT DESCRIPTOR BLOCK

BIT	MASK	
3	10	AS.DEL - TASK HAS DELETE ACCESS (1=YES)
2	4	AS.EXT - TASK HAS EXTEND ACCESS (1=YES)
1	2	AS.WRT - TASK HAS WRITE ACCESS (1=YES)
0	1	AS.RED - TASK HAS READ ACCESS (1=YES)

ATTACHMENT DESCRIPTOR STATUS BITS

Figure 6-15   The Attachment Descriptor Block
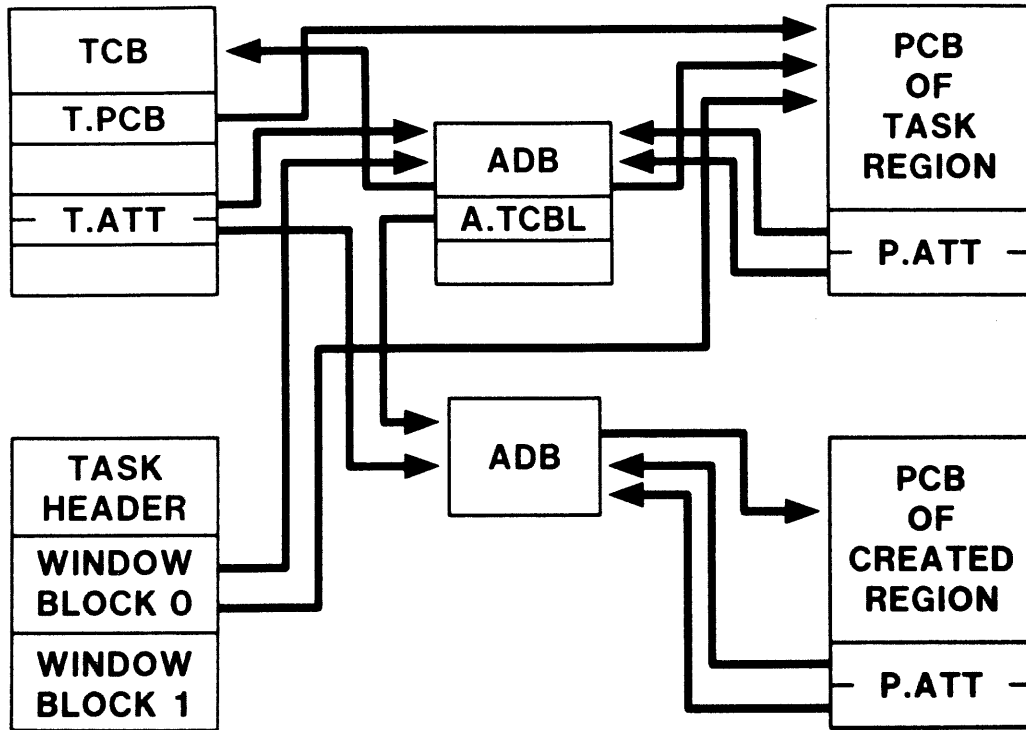
Figure 6-16 Attaching a Task to a Region

Mapping

- Task virtual address window block contains information needed to load APRs to allow access to physical region

- Task is mapped to a region if a window block contains the needed access information for the region

    Pointers to the PCB and ADB for the region

    High and low virtual address limits for the region

    Window size and offset in the physical region in 32 word blocks

    Number of APRs needed for the window and PDR information

POINTER TO NUMBER OF WINDOW BLOCKS	H.WND
NUMBER OF WINDOW BLOCKS	
PCB ADDRESS	W.BPCB
LOW VIRTUAL ADDRESS LIMIT	W.BLVR
HIGH VIRTUAL ADDRESS LIMIT	W.BHVR
ADB ADDRESS	W.BATT
WINDOW SIZE (32 WORD BLOCKS)	W.BSIZ
OFFSET INTO PARTITION (32 WORD BLOCKS)	W.BOFF
NUMBER OF PDRs / FIRST PDR ADDRESS	W.BNPD/W.BFPD
CONTENTS OF LAST PDR	W.BLPD

WINDOW BLOCK 0
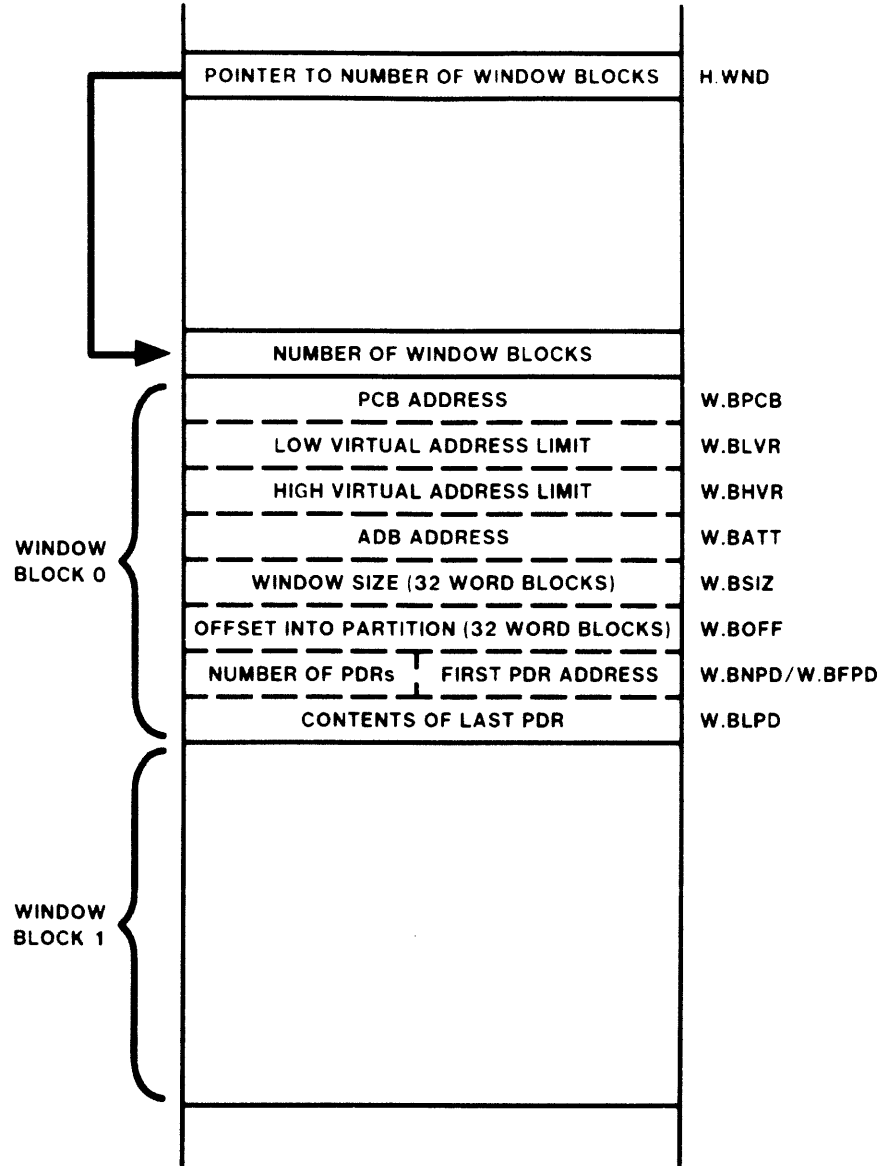
WINDOW BLOCK 1

Figure 6-17   Task Virtual Address Window Blocks
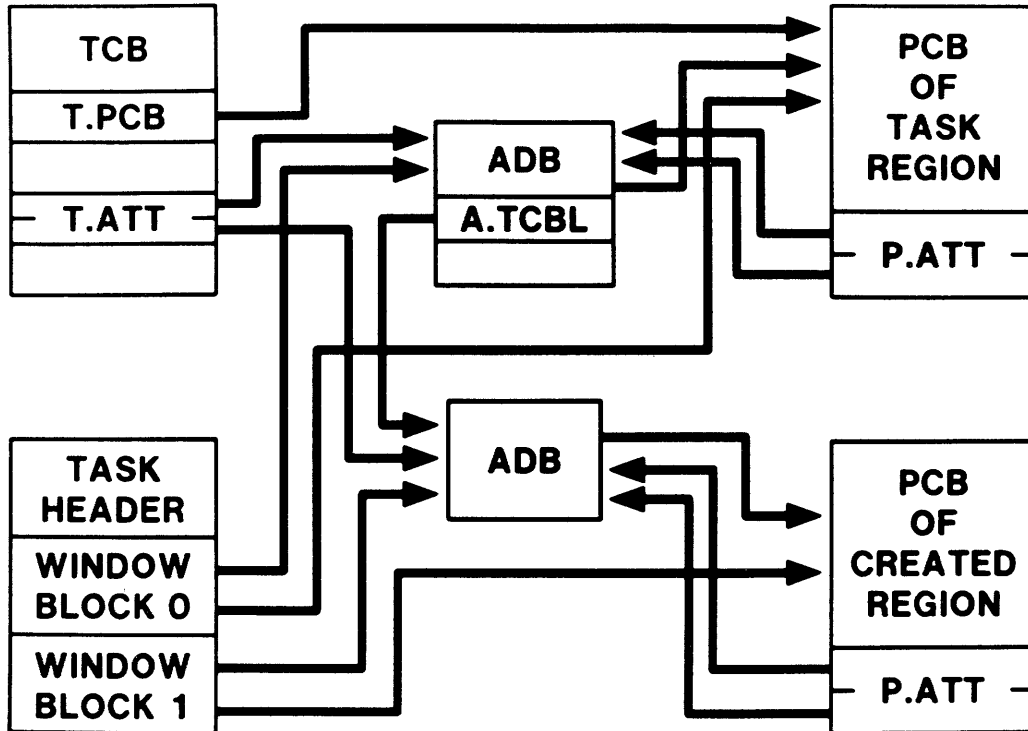
Figure 6-18   Mapping a Task Address Window to a Region

TASK MANAGEMENT

INTRODUCTION

The basic unit of executable code under RSX-11M is the task. Most tasks are user written, but some are system-supplied, such as the task-loader, MCR, etc. The Executive must manage the task scheduling and allocate needed resources to requesting tasks. This module describes the techniques used to manage tasks.

OBJECTIVES

1. Describe the content and functions of the task label blocks and task header.

2. Diagram the data structures used to manage task scheduling.

3. Describe the process of task termination.

RESOURCES

1. RSX-11M System List and Data Structures

2. RSX-11M/M-PLUS Executive Reference Manual

THE TASK IMAGE

The Task Image File

- Created by the taskbuilder

- Includes: label block group, checkpoint area (optional), task header, and task image

The Label Block Group

- Label block Ø

    Task and resident library data

    Used to set up TCB and task addressing windows in header

- Label block 1

    Table of LUN assignments for LUN 1 through LUN 128

    Used to create Logical Unit Table(LUT) in header

- Label block 2

    Continuation of Table of LUN assignments for LUN 129 through LUN 255 (if needed)

    Used to create LUT in header

- Label block 3

    Segment load list (optional)

    Resident libraries that contain memory resident overlays

The Checkpoint Area

- Used to provide checkpoint space when no space available in the system checkpoint files

- Size equal to size of header plus the task image

LABEL BLOCK GROUP		LABEL BLOCK 0 - TASK AND RESIDENT LIBRARY DATA
		LABEL BLOCK 1 - TABLE OF LUN ASSIGNMENTS
		LABEL BLOCK 2 - TABLE OF LUN ASSIGNMENTS (OPTIONAL)
CHECK POINT AREA		LABEL BLOCK 3 - SEGMENT LOAD LIST (OPTIONAL)
		TASK HEADER - FIXED PART
		TASK HEADER - VARIABLE PART
TASK HEADER		ROOT SEGMENT
		(CONTIGUOUS BLOCKS)
		SEGMENT TABLE
TASK IMAGE	OVERLAY SEGMENTS (TASK-RESIDENT OVERLAY DATA BASE)	AUTOLOAD VECTOR
		REGION DESCRIPTOR
		WINDOW DESCRIPTORS
		SEGMENT DESCRIPTORS

Figure 7-1   Task Image File

The Task Header

- Consists of two parts: fixed and variable

- Fixed part contains

  Event Flag mask word and address

  Initial values for PS, PC and SP

  Low-Core context: $DSW, and inpure storage area addresses for FCS, FORTRAN, etc.

  Pointers to specific areas within the variable part

- Variable part contains

  Window blocks

  Logical unit table

  Task context

- Window Block

  Describes task mapping

  Associates window with region

     Contains pointer to PCBs of regions used by task
     Pointers loaded at task installation

  Used to load APRs for task

  Initialized when task is installed

- Logical Unit Table

  Contains a two word entry for each logical unit used by the task

     Word one: address of the UCB of the device assigned to the logical unit (UCB address loaded at task installation)

     Word two: address of the FILES-11 window block of an open file

The Task Image

● Task code

● Overlay information

     Autoload vectors

     Region and window descriptors for memory-resident overlays

     Segment descriptors

ADDRESS OF EVENT FLAG CONTEXT	40	H.EFSV
ADDRESS OF FLOATING-POINT CONTEXT	42	H.FPSA
POINTER TO NUMBER OF WINDOW BLOCKS	44	H.WND
DIRECTIVE STATUS WORD	46	H.DSW
ADDRESS OF FCS IMPURE STORAGE	50	H.FCS
ADDRESS OF FORTRAN IMPURE STORAGE	52	H.FORT
ADDRESS OF OVERLAY IMPURE STORAGE	54	H.OVLY
ADDRESS OF IMPURE VECTORS	56	H.VEXT
MAILBOX LUN / SWAPPING PRIORITY	60	H.SPRI/H.NML
UNUSED	62	
RESERVED	64	
RESERVED	66	
RESERVED	70	
HEADER GUARD WORD POINTER	72	H.GARD
NUMBER OF LUNS	74	H.NLUN

TASK HEADER - FIXED PART

TASK HEADER - VARIABLE PART

LOW-CORE CONTEXT (46 H.DSW – 56 H.VEXT)

Figure 7-2   The Task Header: Fixed Part

H.LUN	LUN TABLE (2 WORDS PER LUN)	
	⋮	

		OFFSETS
NUMBER OF WINDOW BLOCKS		
PARTITION CONTROL BLOCK ADDRESS		W.BPCB
LOW VIRTUAL ADDRESS LIMIT		W.BLVR
HIGH VIRTUAL ADDRESS LIMIT		W.BHVR
ADDRESS OF ATTACHMENT DESCRIPTOR		W.BATT
WINDOW SIZE (IN 32-WORD BLOCKS)		W.BSIZ
OFFSET INTO PARTITION (IN 32-WORD BLOCKS)		W.BOFF
NUMBER OF PDRS TO MAP	FIRST PDR ADDRESS	W.BNPD/W.BFPD
CONTENTS OF LAST PDR		W.BLPD

TASK HEADER - FIXED PART

TASK HEADER - VARIABLE PART

Figure 7-2   The Task Header: Beginning of Variable Part

TASK HEADER - FIXED PART

TASK HEADER - VARIABLE PART

CURRENT PS	
CURRENT PC	INITIAL VALUES
CURRENT R5	RELATIVE BLOCK NUMBER OF HEADER
CURRENT R4	IDENT. WORD #2
CURRENT R3	IDENT. WORD #1
CURRENT R2	TASK NAME WORD #2
CURRENT R1	TASK NAME WORD #1
CURRENT R0	PROGRAM TRANSFER ADDRESS
HEADER GUARD WORD	
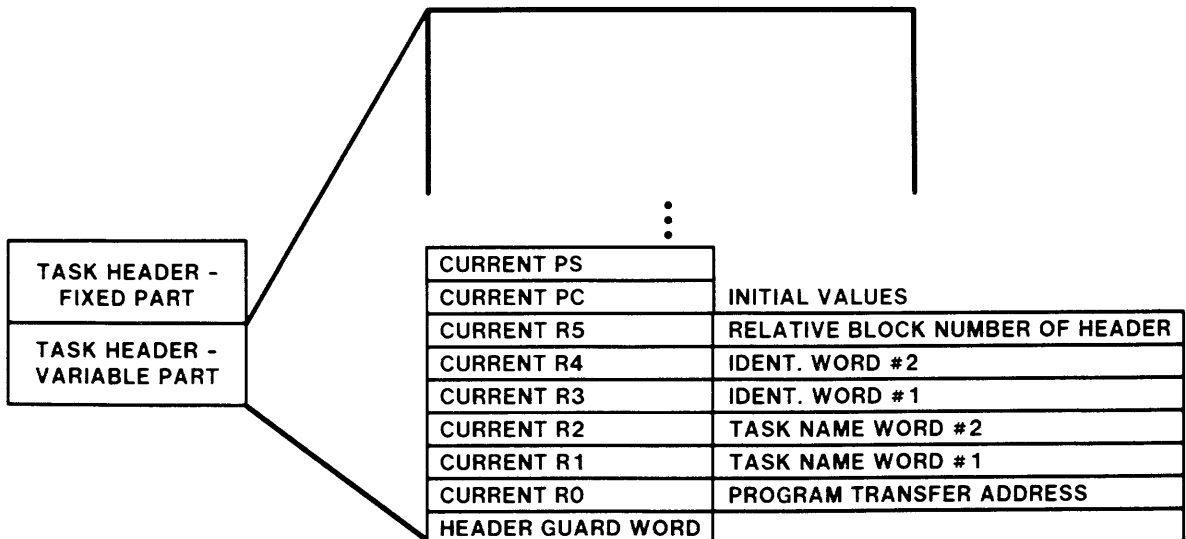
Figure 7-2   The Task Header: End of Variable Part

169

RUNNING A TASK

- Performed by MCR RUN command

    Directives

        RUN$
        RQST$
        SPWN$

- If task not installed, use MCR command

    Task is installed with name equal to terminal, TTnn

    Task is executed

    Task is removed on exit

- Two types of requests:

    Immediate request - is executed as soon as possible

    Time-dependent request

        Executed at a specified time

        Rescheduled to run periodically

- Steps in running a task

    Task is installed

    Task is activated

    Memory is allocated to task

    Task is loaded

    Task is scheduled

    Task is terminated

Installing

- Task is installed by MCR command

    Permanent installation by INS command

    Temporary installation by RUN command

- TCB is created and initialized

- Location T.PCB in TCB points to:

    The PCB of the partition or subpartition for which it was built (if partition is user-controlled)

    The PCB of the main partition (if partition is system-controlled)

- Locations T.LDV and T.LBN point to the load device and logical block number of the task image file

- Access rights to resident commons and libraries are checked

- Task TCB is inserted into the STD by routine $QINSP in module QUEUE

- Window blocks in header are initialized from the label blocks

- LUT is initialized from the label blocks and UCB addresses are loaded

Activating

- Task placed in the Active Task List (ATL)

- If task not installed and not privileged, it is temporarily installed

Allocating Memory

- Partition wait-queue used to schedule memory use

    Ordered by task priority

    Linked using T.LNK, the utility link word

    Starts at P.WAIT in PCB

- TCB placed in partition wait-queue

    When task is requested

    When task is checkpointed

Loading

- Task placed in loader queue

    Utility link word, T.LNK, links TCBs in queue

    Location P.WAIT in PCB is queue listhead

- In mapped system, task header copied into DSR

    Executive updates copy in DSR

    $DSW only data in task copy that is updated
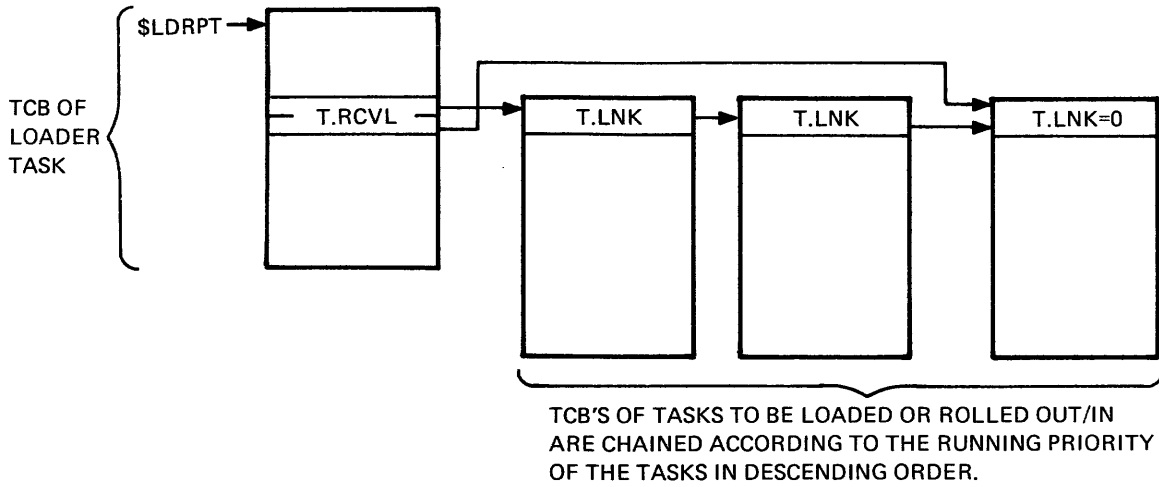
Figure 7-3   The Loader Queue

Context Switching

- Current task's context stored in task header

- Task context includes:

    General registers (R0 - R5)

    Processor Status Word

    Stack pointer (R6)

    Program counter (R7)

    Floating-point registers

- Mapping context for current task is not saved

- New task's context is loaded from its header

- Mapping context for new task created using window blocks and PCBs
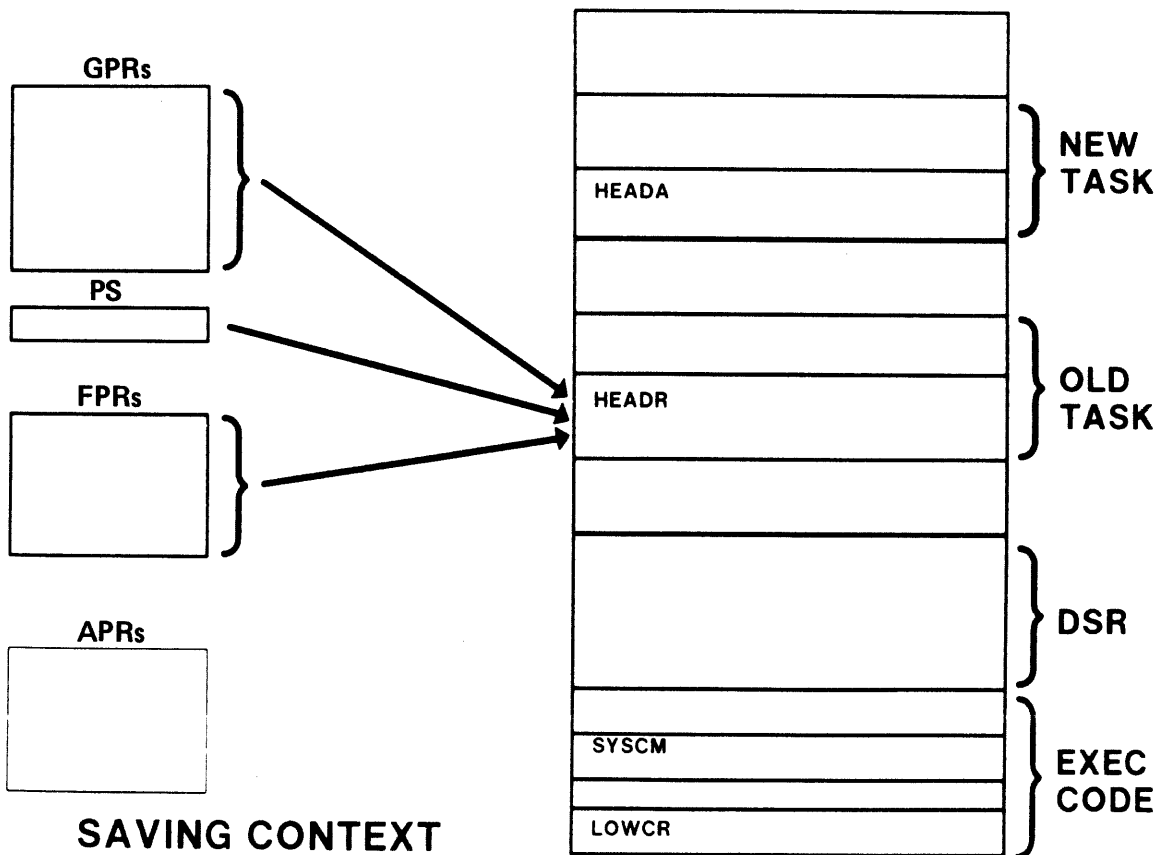
Figure 7-4   Context Switching: Saving Context

174

GPRs

PS

FPRs

APRs

LOADING CONTEXT

NEW TASK

HEADA

WINDOW BLOCKS

OLD TASK

HEADA

DSR

TCB OF
NEW TASK

EXEC CODE

SYSCM
$TKTCB::

LOWER
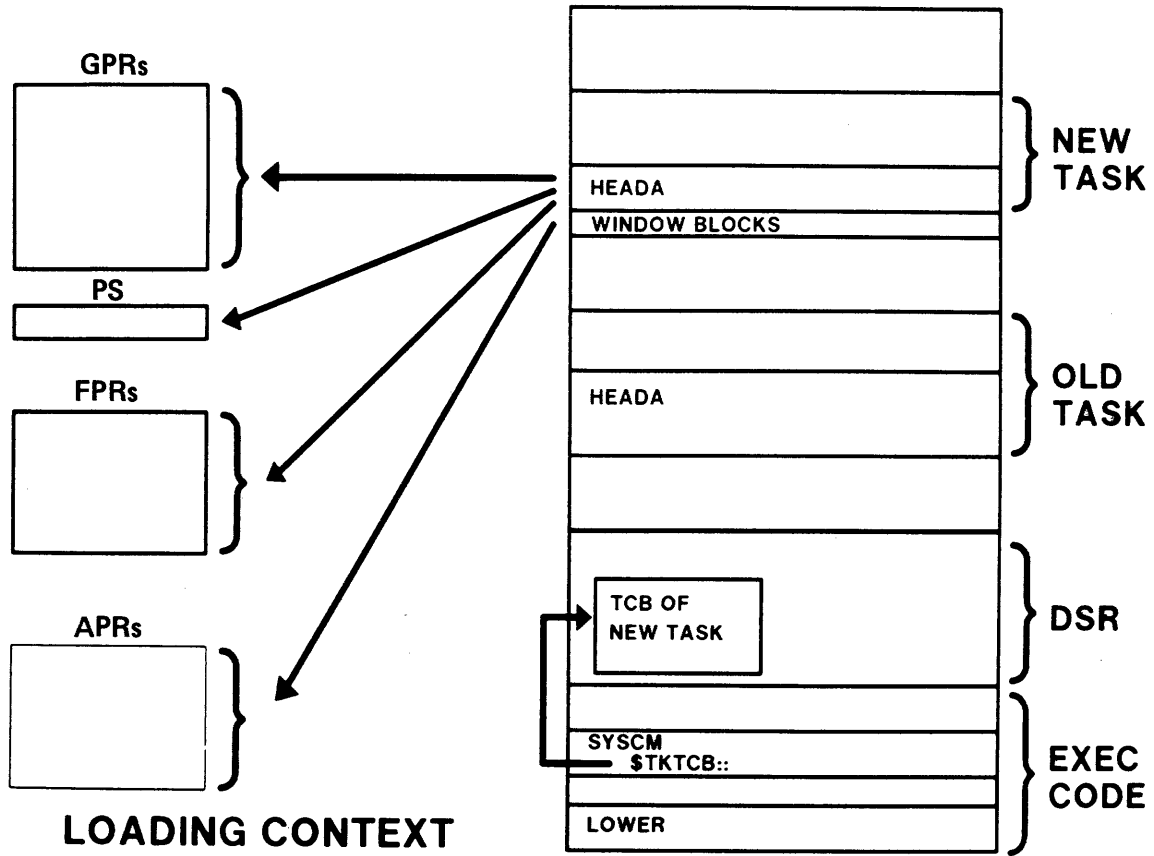
Figure 7-4   Context Switching: Loading Context
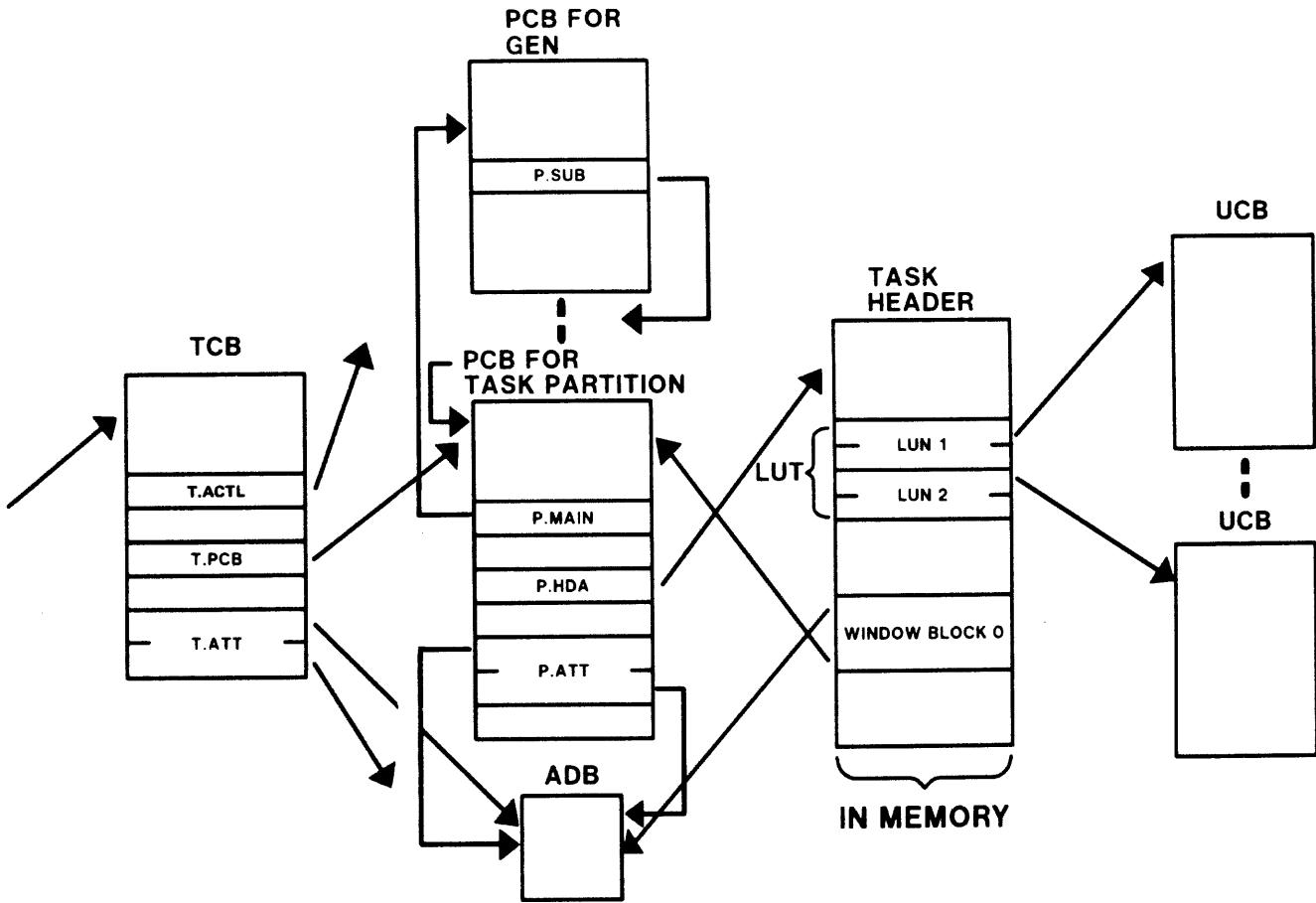
Figure 7-5   The Data Structures for an Active Task

SCHEDULING A TASK

Priority

- Scheduling request sets value in $RQSCH

- Value can be:

    The beginning of the ATL (e.g., after a significant event)

    Address of particular TCB (e.g., after a conditional scheduling request)

- Scheduling algorithm scans the ATL beginning at the TCB whose address is in $RQSCH

- System chooses:

    First resident unblocked task

    First task with a pending AST

Round Robin

- Tasks of the same priority share the CPU

- Is a SYSGEN option

- Parameters set at SYSGEN

    Round robin scheduling time interval

    Priority range for round robin

- For each priority in priority range

    Sublist of tasks of that priority is scanned

    TCBs of blocked, suspended or waiting tasks are skipped

    TCB of first eligible task is placed last in list of tasks with same priority
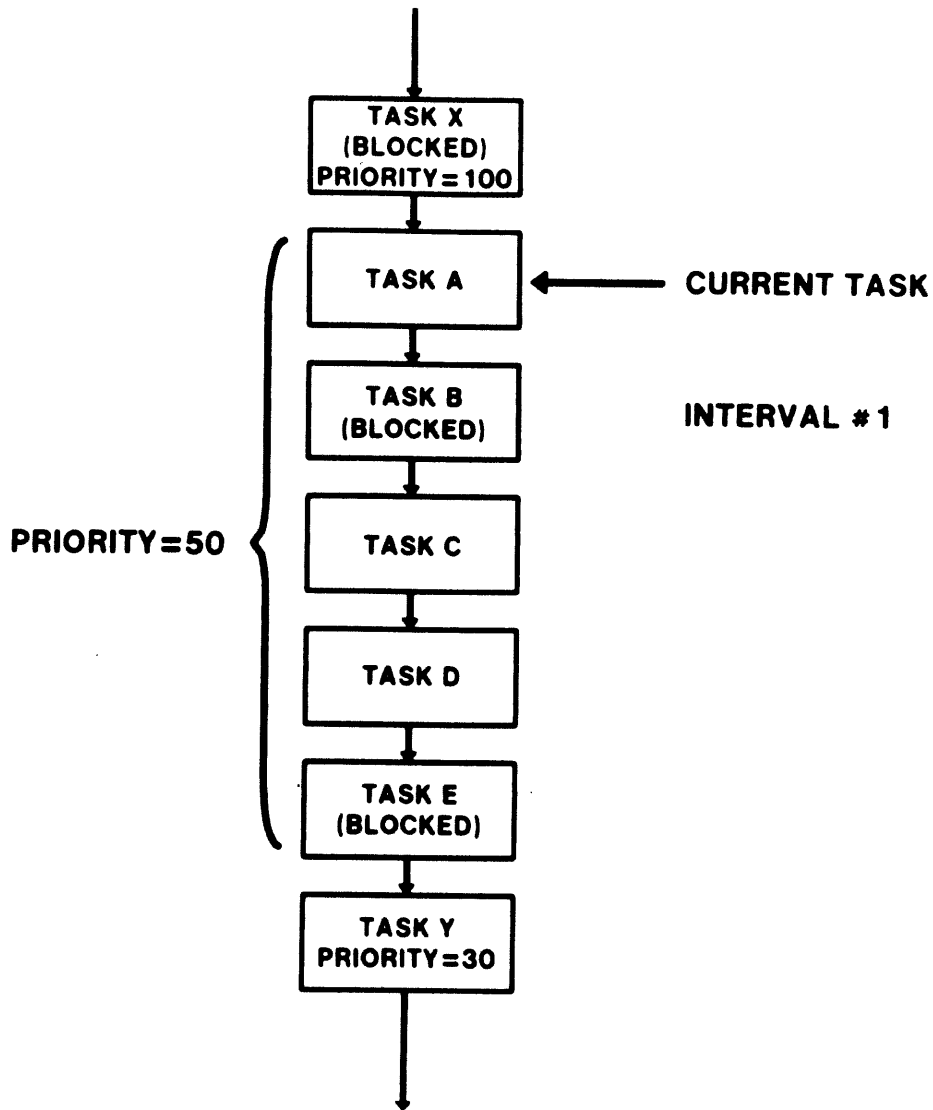
Figure 7-6   Round Robin Scheduling: Interval #1
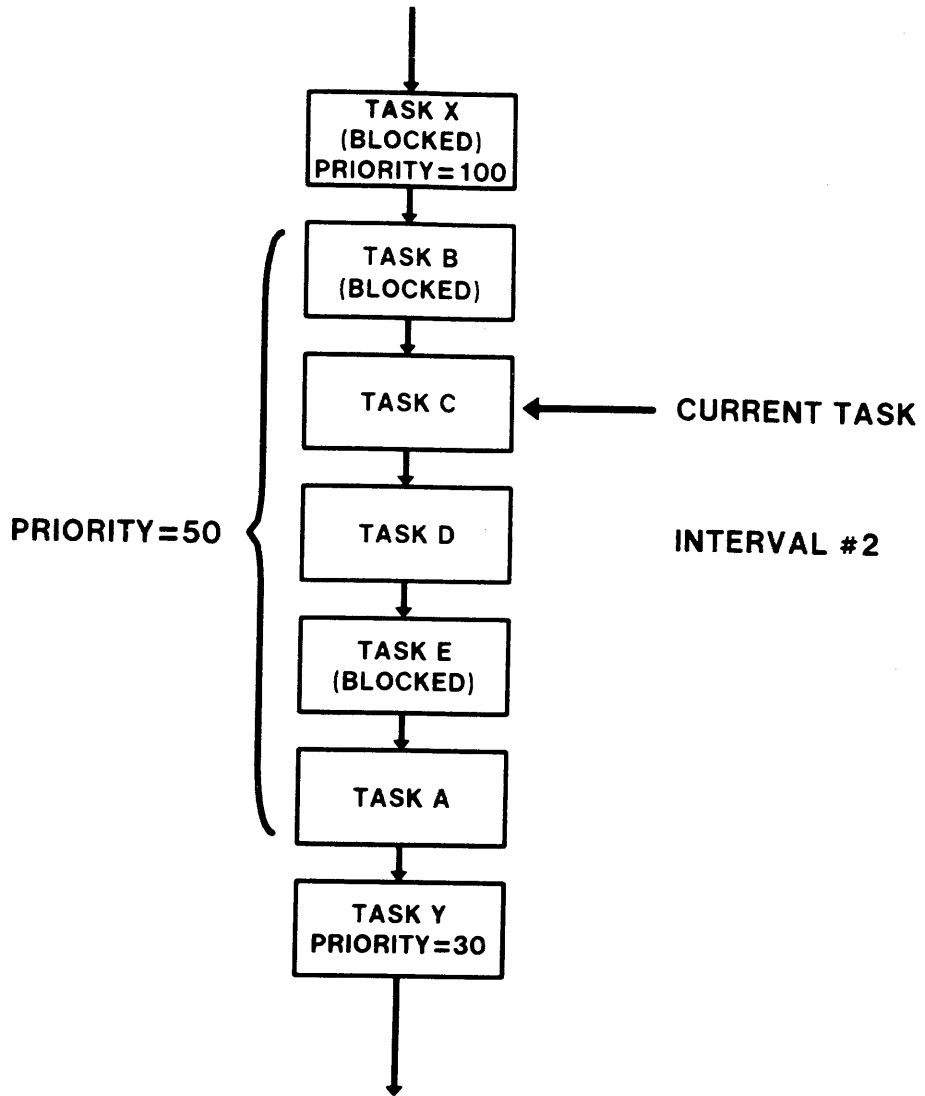
Figure 7-6   Round Robin Scheduling: Interval #2

Disk Swapping

● Allows tasks of similar priority to share memory

● Is a SYSGEN option

● Parameters set at SYSGEN

    Swapping priority

    Time interval for swapping

● Steps in disk swapping

    When task loaded, swapping priority is set at H.SPRI in the task header

    At the end of each swapping interval, H.SPRI is decremented

    H.SPRI is not decremented beyond the negative of the swapping priority

    For checkpointing, H.SPRI is added to the running priority

    When task is checkpointed, H.SPRI is set back to the swapping priority

Clock Scheduling

- Provides facility for tasks to run

    At specific times

    After specific time intervals

- When clock interrupt occurs, the clock queue is scanned

- Types of time dependent requests

    Mark time request

    Task request with periodic rescheduling

    Single shot task request

    Single shot internal subroutine
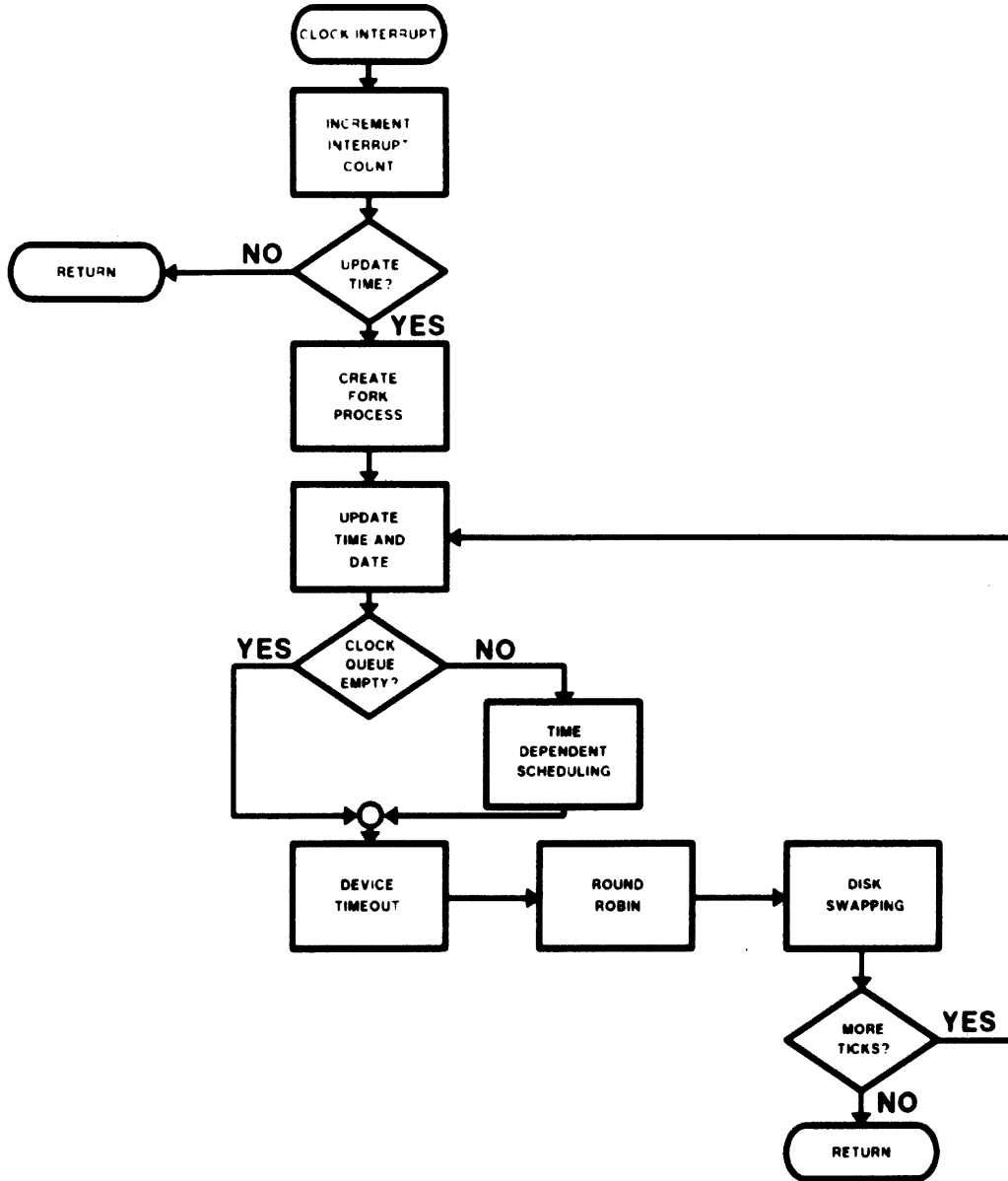
Figure 7-7   Time-Dependent Scheduling

AST Processing

- AST is used to notify a task when an asynchronous event has occurred

- Three phases in processing an AST

  Specifying the AST

  Allocate an AST control block

  Fill in the type of AST at A.CBL

  Link the block to the TCB at T.SAST

  Declaring the AST

  Queue the AST control block to the AST queue at T.ASTL

  Scheduling the AST

  Conditional schedule request is made

  After context switch to task, RESCH in SYSXT attempts to dequeue an AST

  Stopped task is activated to process the AST, then stopped again

- When an AST is scheduled

  Necessary data is pushed onto the task stack

  AST executes before return to task code
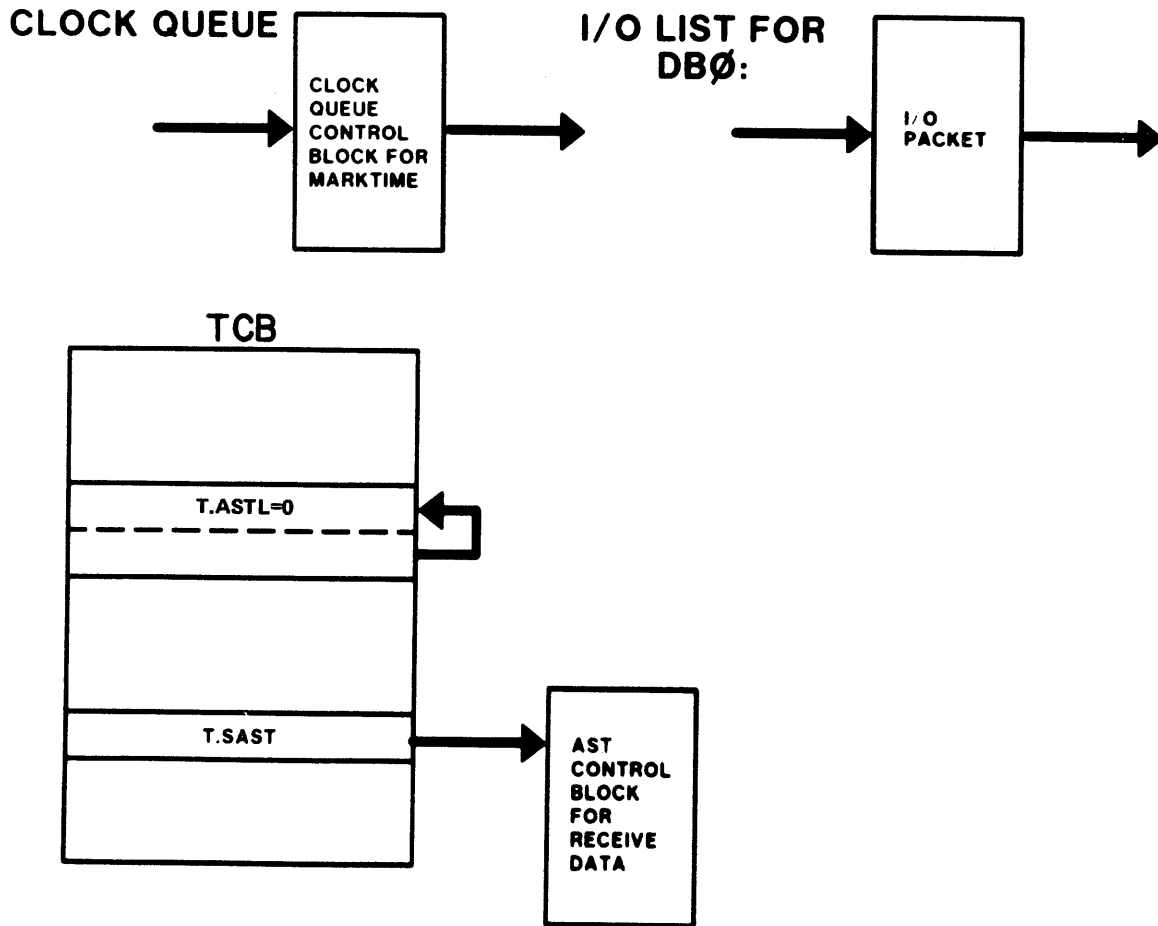
Figure 7-8   AST Processing: Sources of ASTs

CLOCK QUEUE

I/O LIST FOR DBØ:

TCB

CLOCK QUEUE CONTROL BLOCK FOR MARKTIME

I/O PACKET

T.ASTL

AST CONTROL BLOCK FOR MARKTIME

T.SAST

AST CONTROL BLOCK FOR RECEIVE DATA
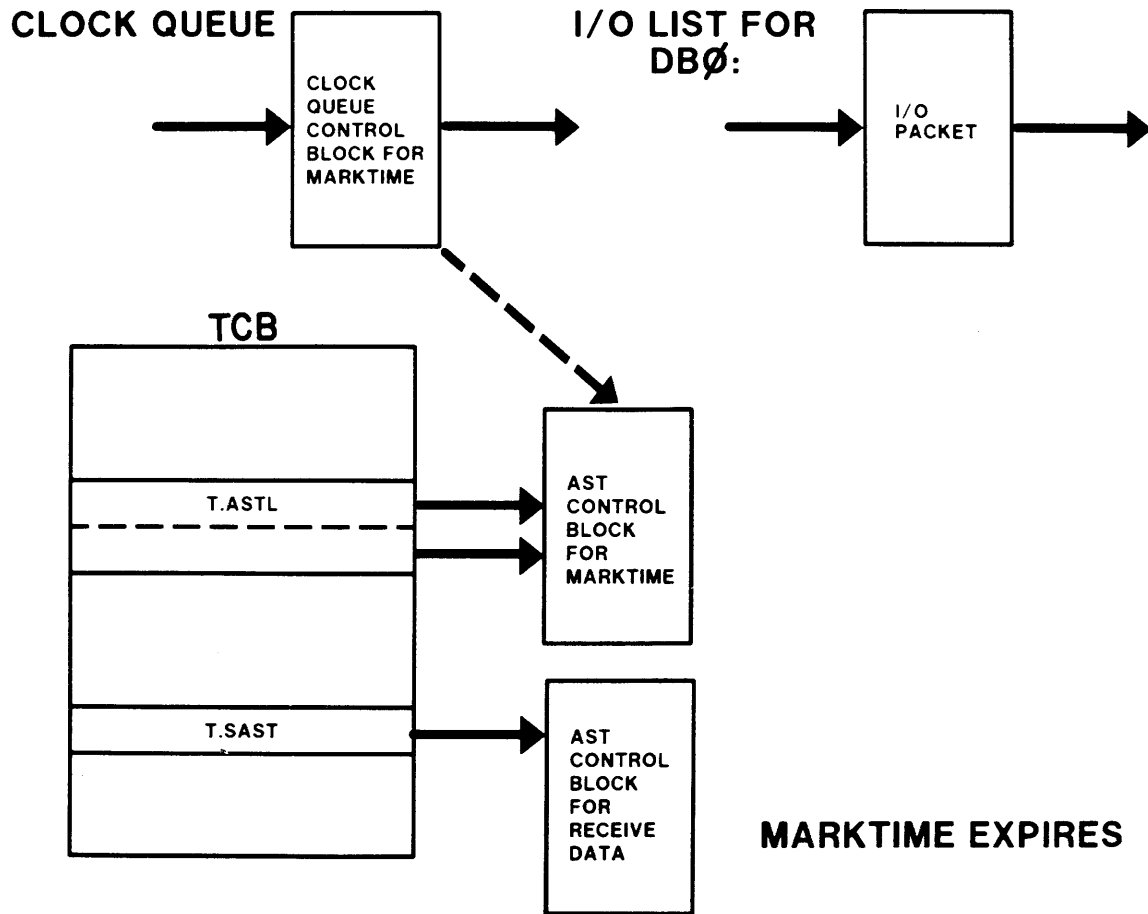
MARKTIME EXPIRES

Figure 7-8   AST Processing: Marktime Expiration

I/O LIST FOR DBØ:

DATA RECEIVED BY TASK
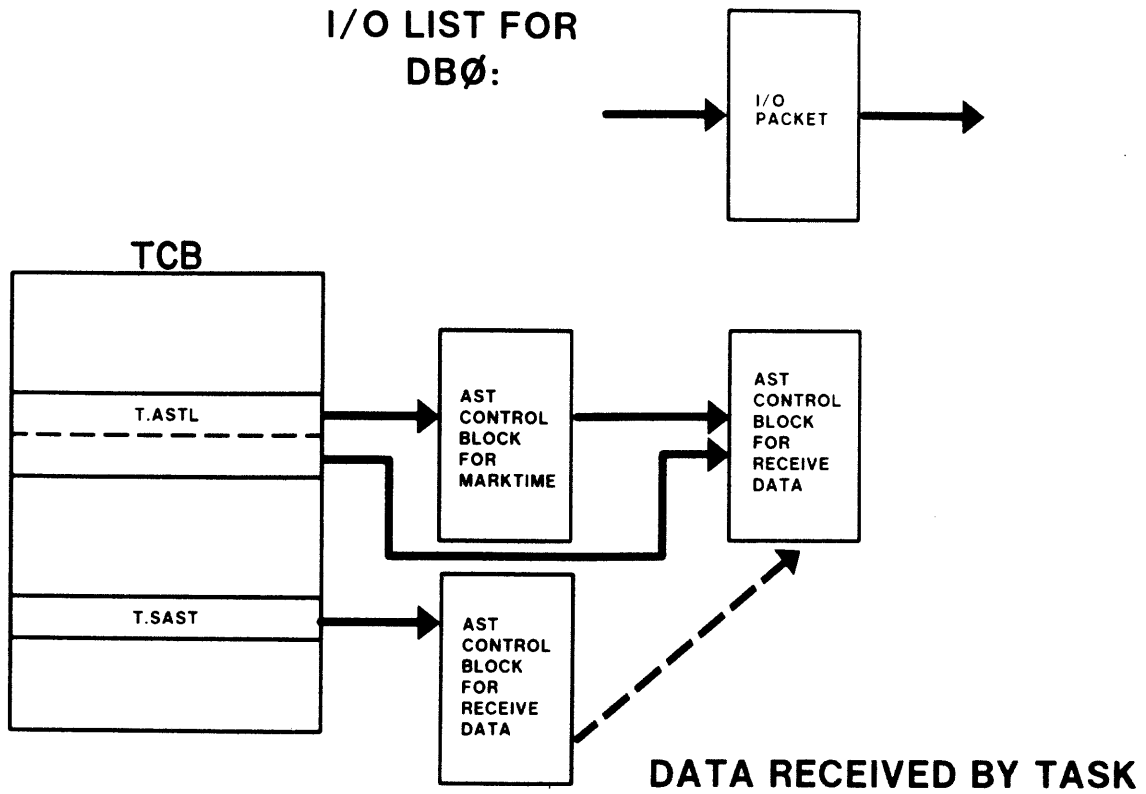
Figure 7-8   AST Processing: Completion of a Receive Data

I/O LIST FOR DBØ:

I/O REQUEST COMPLETES
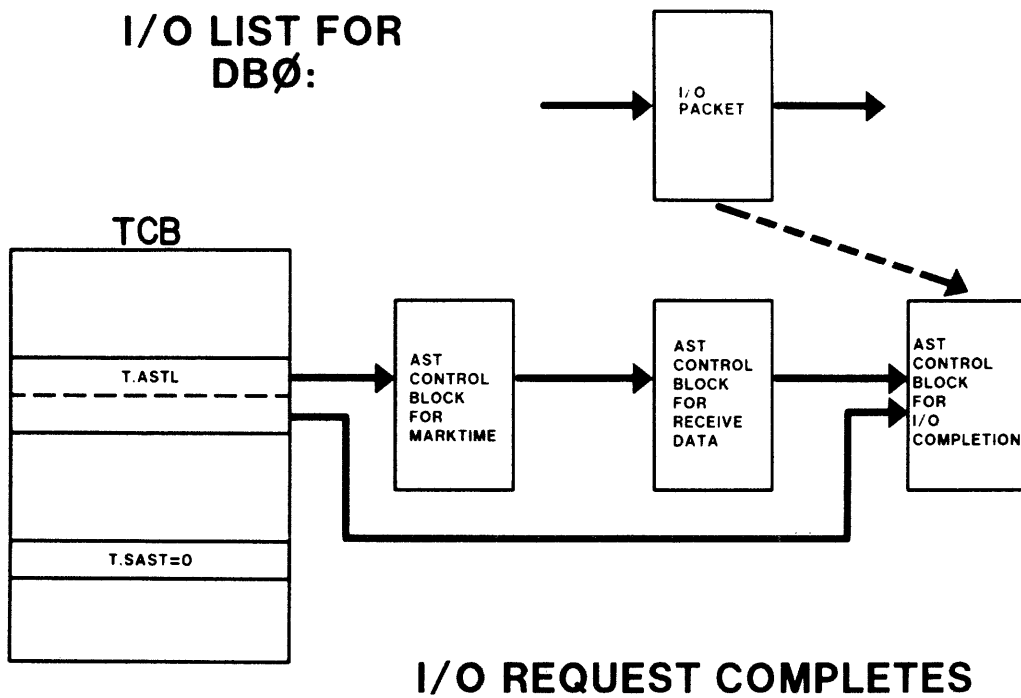
Figure 7-8   AST Processing: Completion of an I/O Operation

TASK TERMINATION

- Normal termination

    Task issues an exit directive

- Abnormal termination

    Another task issues an abort directive

    Task is aborted by an MCR abort command

    Task is aborted because of an SST

- Actions performed at task termination

    Halt task

    Cleanup system database

    Cancel mark time requests

    Break links to offspring tasks

    Disconnect interrupt vectors, if any

    If T3.REM = 1, deallocate TCB

    Deallocate header in pool if mapped system

    If abnormal exit, run TKTN

    Declare a significant event

- I/O Rundown

    Kill all I/O on each assigned LUN

    Detach all attached devices

    Close all files opened by task

Synchronous System Traps

- Trap vector for each type

- System response to SST

        Kernel mode: Crash system

        User mode: Abort task if no user SST vector

                          Execute user-supplied routine if user SST vector

Table 7-1 Synchronous System Traps

Trap Vector	Trap Routine	Service Routine	Cause
4	$TRP04	$TRP04	Odd or nonexistent memory address error
250	$SGFLT	$SGFLT	Memory protect violation
14	$TRACE	$TRACE	T-bit or BPT instruction
20	$IOTRP	$IOTRP	IOT instruction
10	$ILINS	$ILINS	Reserved instruction
30	$EMTRP	$EMSST	Non-RSX EMT instruction
34	$TRTRP	$TRTRP	TRAP instruction
244	$FPPR8 (or $FPPR7)	$FPPR8	Synchronous floating-point exception
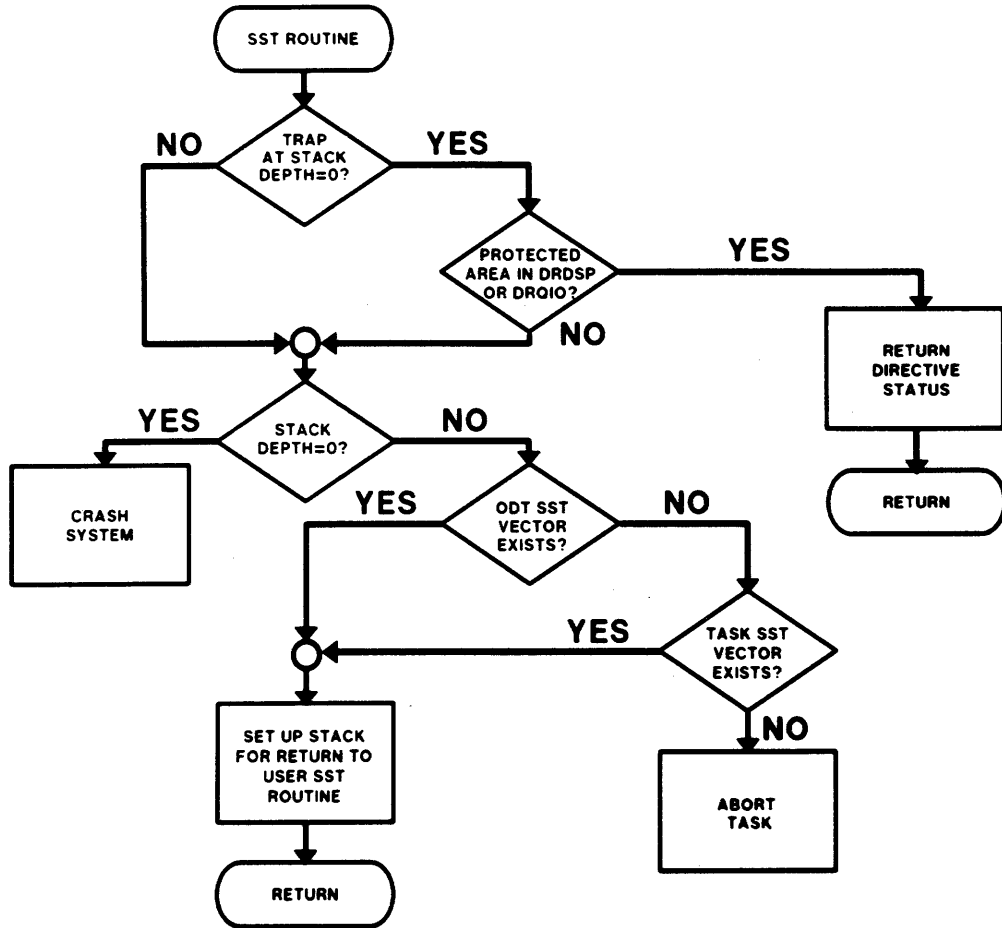
Figure 7-9   SST Processing

8

SYSTEM SYNCHRONIZATION

INTRODUCTION

A large number of activities are in progress at any given time while an RSX-11M system is in use. System synchronization is the term given to the system processing which keeps these activities from interfering with each other. One important activity which must be controlled is access to the system data base.

OBJECTIVES

1. Describe how a software trap is processed by the system.

2. Describe how a hardware interrupt is processed by the system.

3. Describe FORK processing.

4. Describe how a privileged task gets access to the system data base and code.

RESOURCE

1. RSX-11M System Lists and Data Structures

OVERVIEW

- Executive is driven by interrupts and traps

    Idle until interrupt or trap occurs

- System operation

    Executive booted into memory by ROM

    ROM starts Executive by initializing PS and PC

    Executive initializes the environment

        Enables memory management unit

        Turns on clock

        Drivers called at powerfail entry points

    Executive drops into idle loop and waits for interrupts

        Command typed in at a terminal

            Interrupt occurs

            Command passed to MCR or DCL

            MCR or DCL runs a task

            Task may issue traps

        Clock may interrupt functions

System Synchronization Functions

● Provides serial access to system database

 Process accessing system database 'owns' the database until finished

 'Race' conditions are avoided

● Provides orderly method of allocating CPU use between user and system

Software States

● User State

 User code executing

 Processor in USER mode

 Processor fully interruptable – Hardware priority is 0

● System State

 Only state in which the shared system database should be accessed

 Processor in KERNEL mode

 Processor fully interruptable – Hardware priority is 0

● Interrupt Service State

 Used to perform critical operations after an interrupt

 Processor in KERNEL mode

 Processor not fully interruptable – Hardware priority between 4 and 7

Software Processes

- Process - A stream of code in KERNEL mode performing a complete logical job

  Can be in the Executive or in a privileged task

- There are three types of processes contained in the Executive

  Trap process

    Services a software trap

    Performed in system state

  Interrupt Process

    Services a hardware interrupt

    Performed in interrupt service state

  FORK Process

    Allows process or task to get access to system database

    Performed in system state

The Stack Depth Indicator

- One word at location $STKDP in the Executive module SYSCM

- Indicates software state and number of interrupts which have occurred in KERNEL mode

  Value of +1 indicates processing in user state

  Value of 0 indicates processing in system or interrupt service state

  Trap or interrupt occurred from user state

  Negative value indicates processing in interrupt service state

  Trap occurred from system state

- Each time trap or interrupt occur, $STKDP is decreased by 1

- Each time a trap or interrupt process completes, $STKDP is increased by 1

Traps and Interrupts

- CPU hardware performs following steps

  New program counter and processor status are obtained from the vector

  Current program counter and processor status are pushed onto current stack

  Current stack is KERNEL stack if mapped system
  Current stack is USER stack if unmapped system

- New program counter contains address of trap service routine

- New processor status causes

  Processor to change to KERNEL mode
  Processor priority to change to level 7

PHYSICAL MEMORY

$STACK::

SYSTEM STACK

| OLD PS |
| OLD PC |

"CONTEXT SWITCH"

VECTORS

| NEW PS |
| NEW PC |

PC

PS

●●●●●●
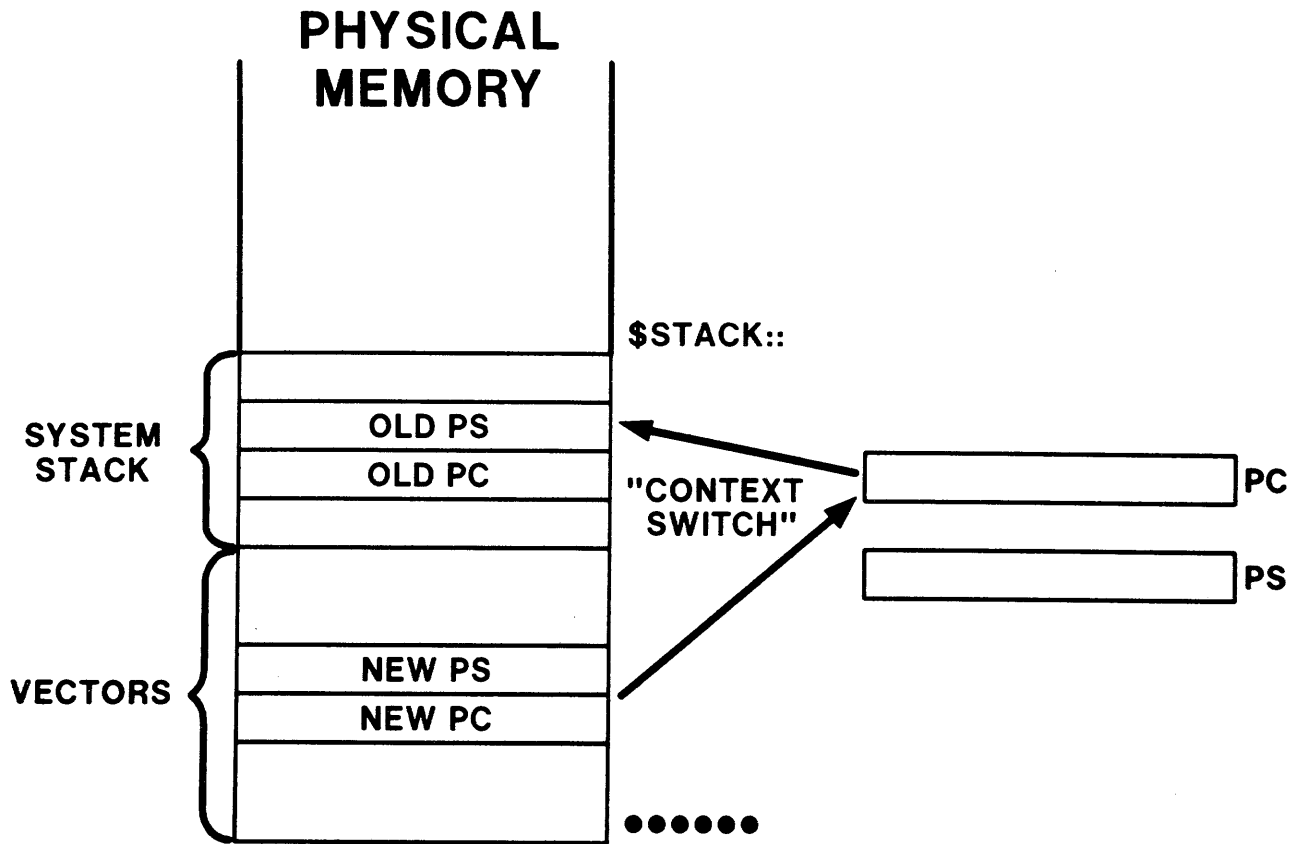
Figure 8-1   Handling Traps and Interrupts by the Hardware

TRAP PROCESSING

1. Trap caused by execution of a 'trap' instruction or synchronous system trap detected by the Executive

2. Each type of trap corresponds to a unique two word vector in low memory

3. Most common trap is 'EMT 377', the result of a directive call

4. EMT instructions

   Cause a transfer to location $EMTRP in module DRSUB

   Processing occurs in module $DRDSP, the Directive Dispatcher
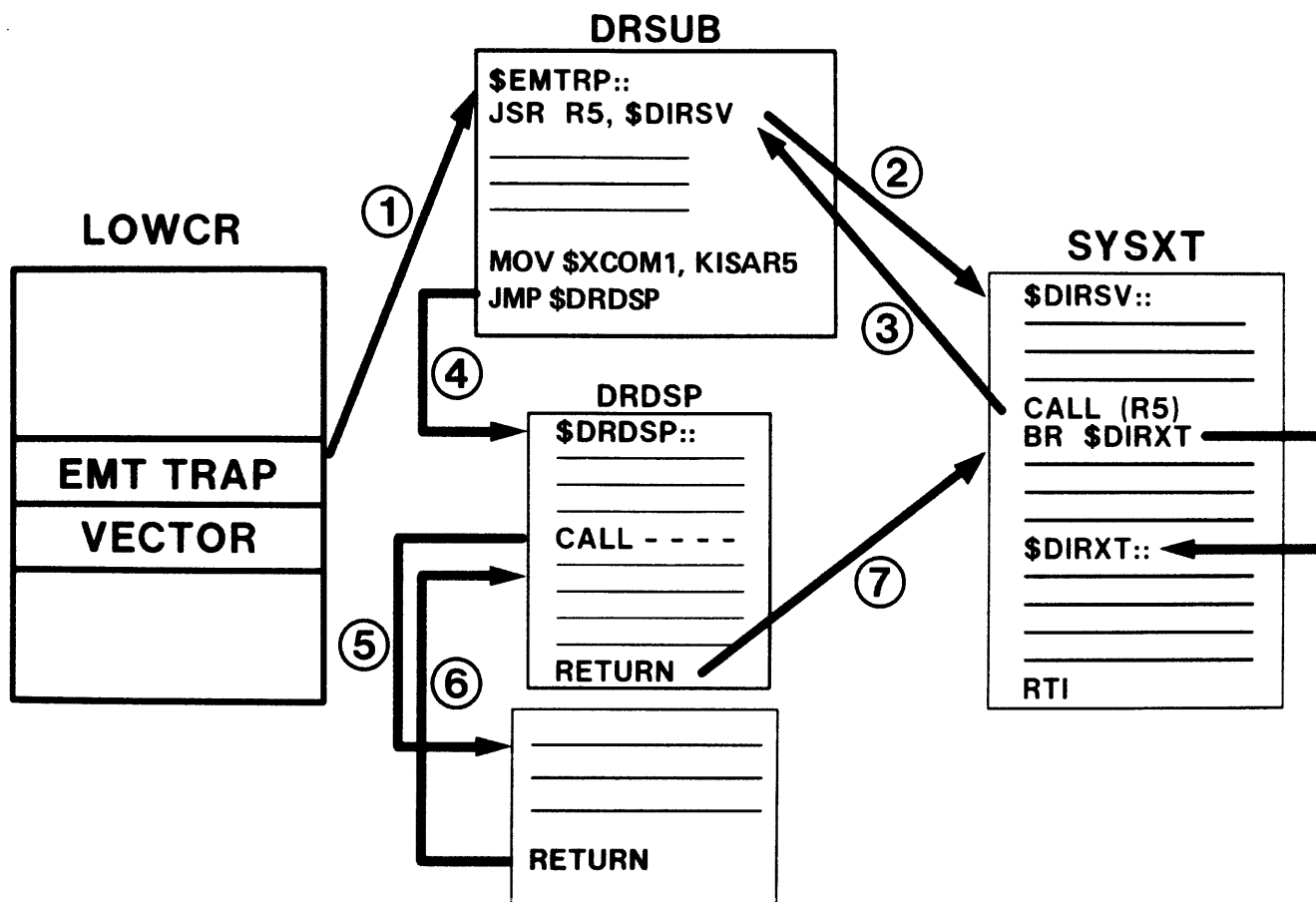
Figure 8-2   EMT Instruction Execution

The Directive Save Routine

- Code begins at label $DIRSV in module SYSXT

- Called by trap processing routine upon entry

- Processing in directive save routine

    Stack depth indicator is decremented to 0

    R5 is saved on current stack by JSR instruction

    R4 is saved on current stack

    On unmapped system, loads R6 with pointer to system stack

    R3, R2, R1 and R0 are saved on system stack

    Performs coroutine call back to DRSUB

The Directive Exit Routine

- Code begins at label $DIRXT in module SYSXT

- Raises processor priority to 7 to lock out interrupts

- Checks for waiting FORK process

    If so: maps driver using KERNEL APR 5
    transfers control to next FORK process

- Increments $STKDP (Should now be +1)

- Restores general purpose registers

- Executes RTI instruction to return to USER mode
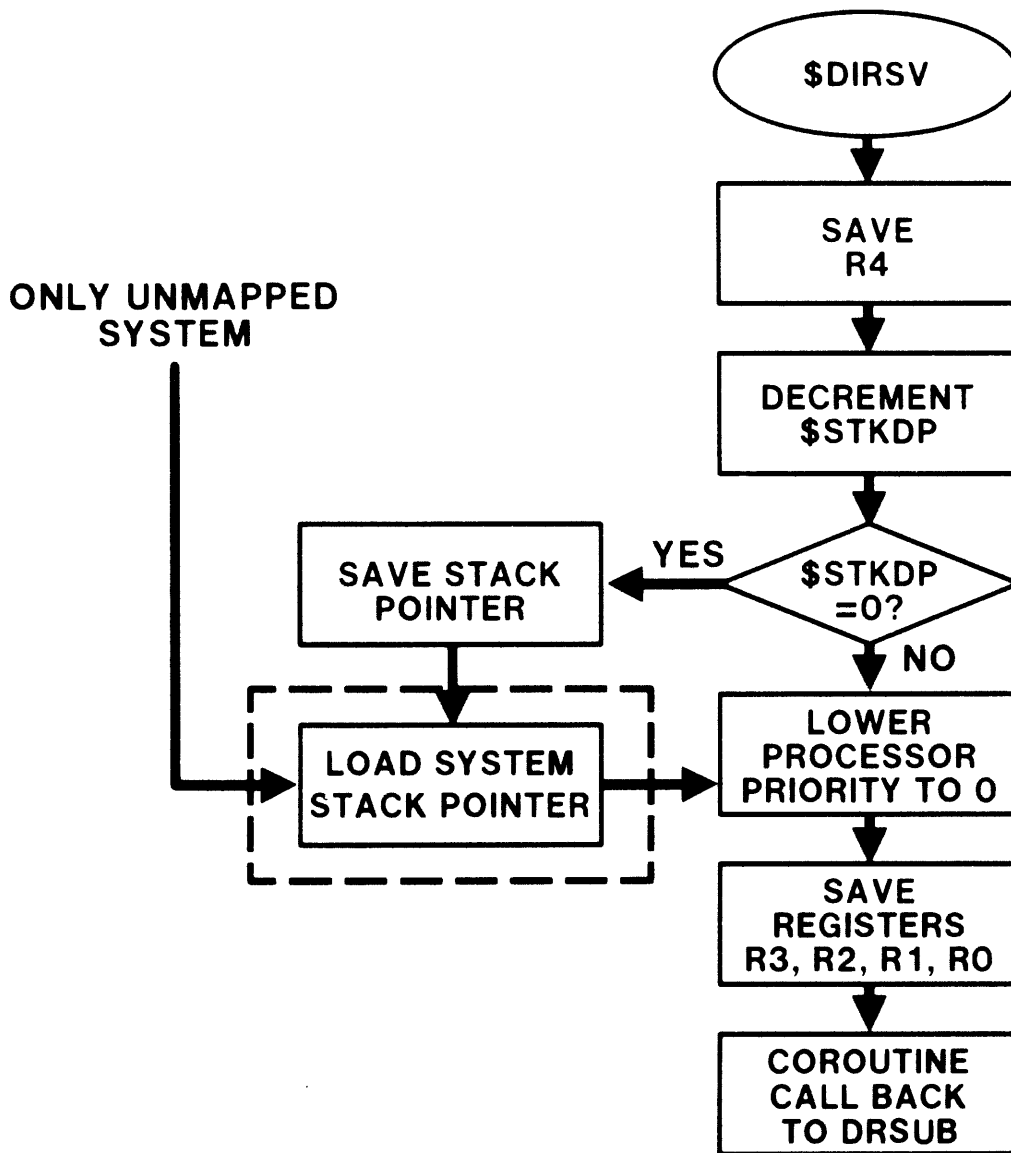
Figure 8-3   The Directive Save Routine
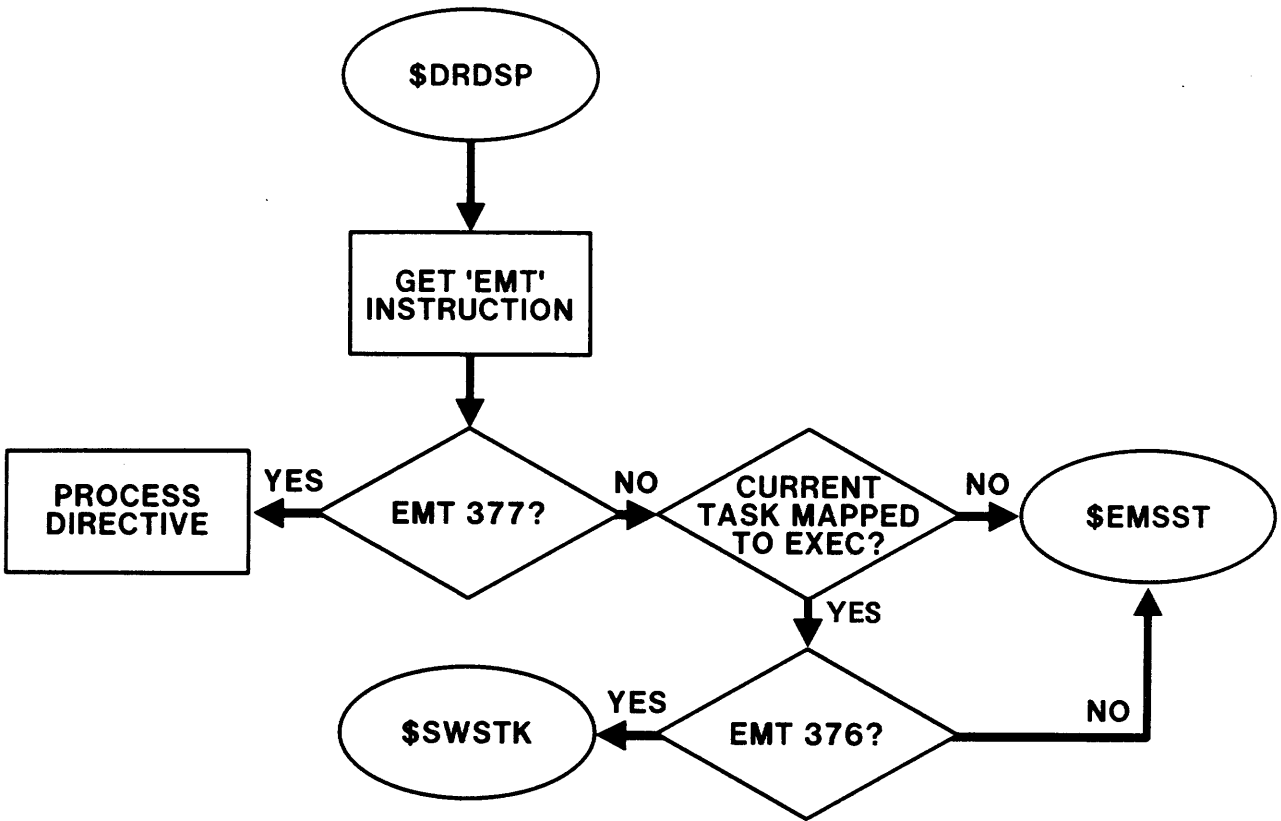
Figure 8-4   Common Processing in the Directive Dispatcher
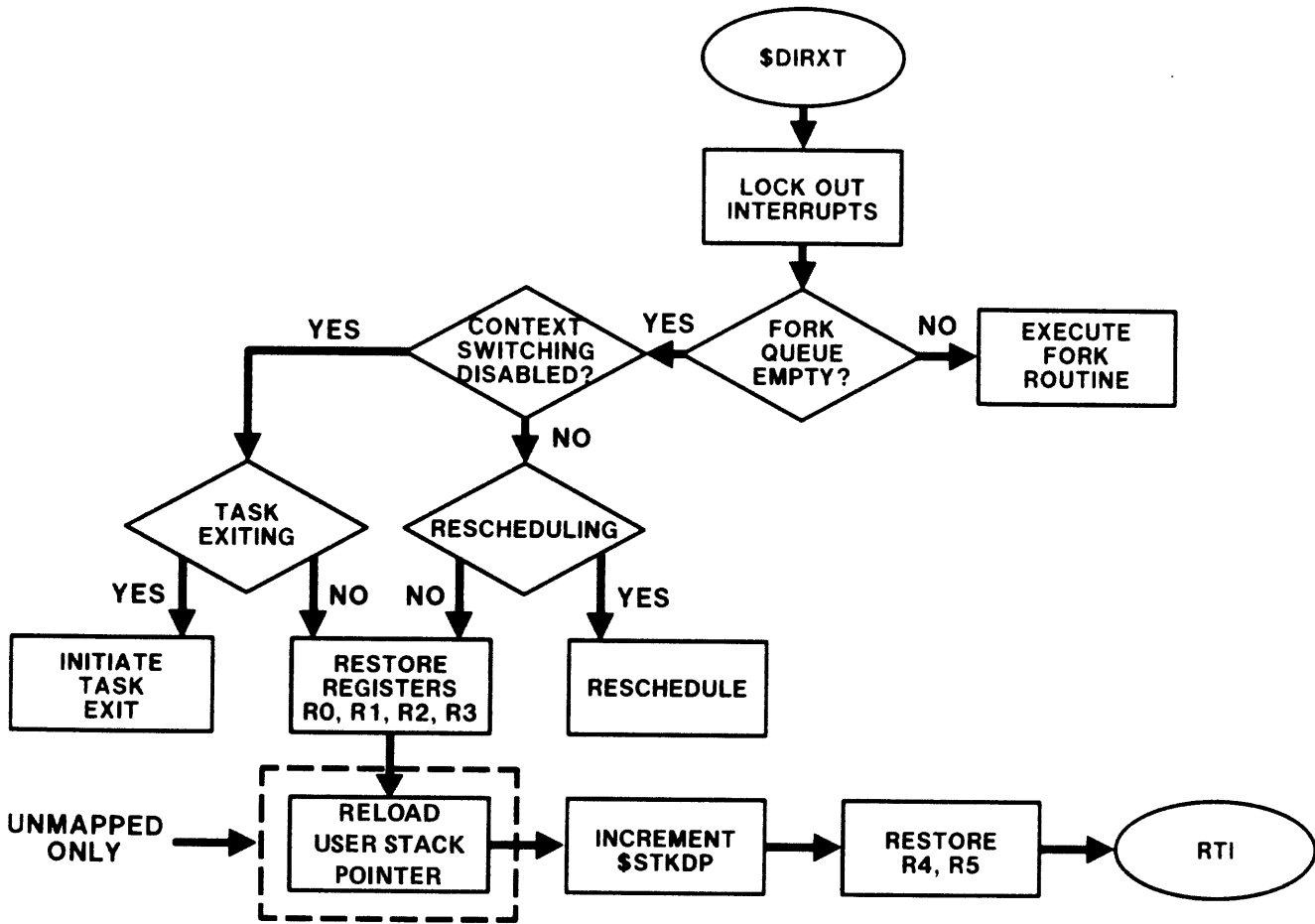
Figure 8-5   The Directive Exit Routine

INTERRUPT PROCESSING

● Interrupt caused by hardware event

● Device driver allows interrupt by setting 'interrupt-enable' bit device register

● Each device controller has its own 2-word interrupt vector in low memory

The Interrupt Save Routine

● Code begins at label $INTSV in module SYSXT

● Entry is with processor priority at level 7 to lock out other interrupts

● Called by interrupt service routine upon entry

● Processing in interrupt save routine

 Stack depth indicator is decremented by one

 R5 is saved on the current stack by JSR R5,$INTSV

 R4 is saved on the current stack

 On unmapped system, SP is loaded with the address of the system stack if entry is from user mode

 Processor priority is lowered to priority of the interrupting device

 Performs coroutine call back to the interrupt service routine

● Several versions of this routine are available in module SYSXT

 $INTSV - Used by devices without error logging

 $INTSE - Used by devices with error logging

 $INTSC - Used by connect-to-interrupt routines
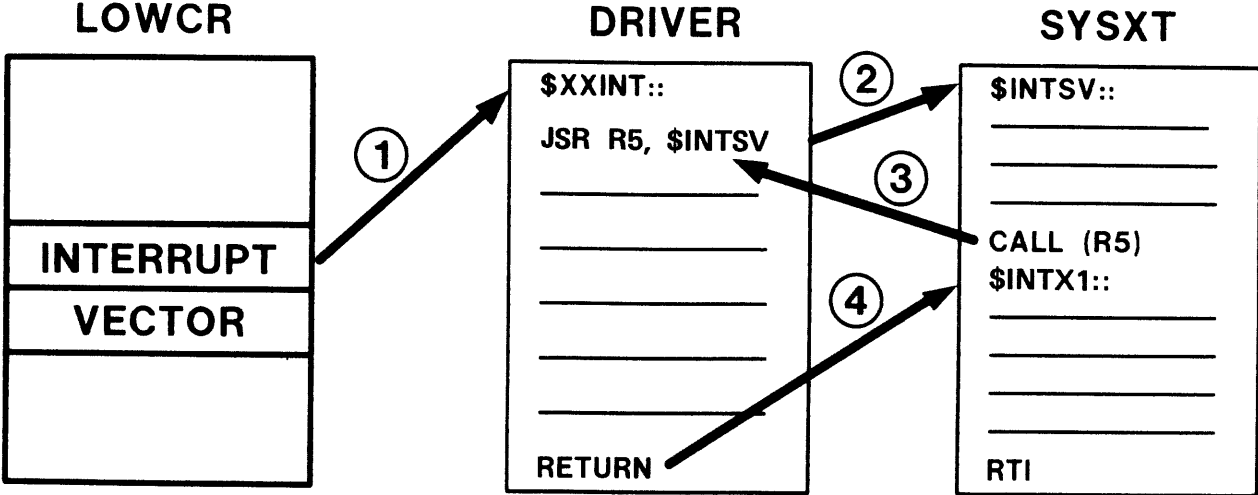
Figure 8-6 Interrupt Processing

Device Driver Processing

- Usually divided into three processing sections

- Processor Status Word in vector causes entry into interrupt service routine at priority 7

  All interrupts are blocked

  Usually $INTSV is called and lowers priority

- Critical processing is performed at the priority of the device

  For a terminal, the last input character is processed and the interrupt-enable bit is set for the next character

  If the system data base must be accessed, a FORK process is created

- A device driver can access the system data base only as a FORK process

  A FORK process executes at processor priority Ø

The Interrupt Exit Routine

- Code begins at label $INTX1 in the module SYSXT

- Raises processor priority to 7 to lock out interrupts

- Checks for waiting FORK process

  If one: lowers priority to Ø
  saves RØ - R3 on system stack
  executes the $DIRXT routine

- Increments $STKDP

- Restores general registers

- Executes RTI instruction to return to USER mode
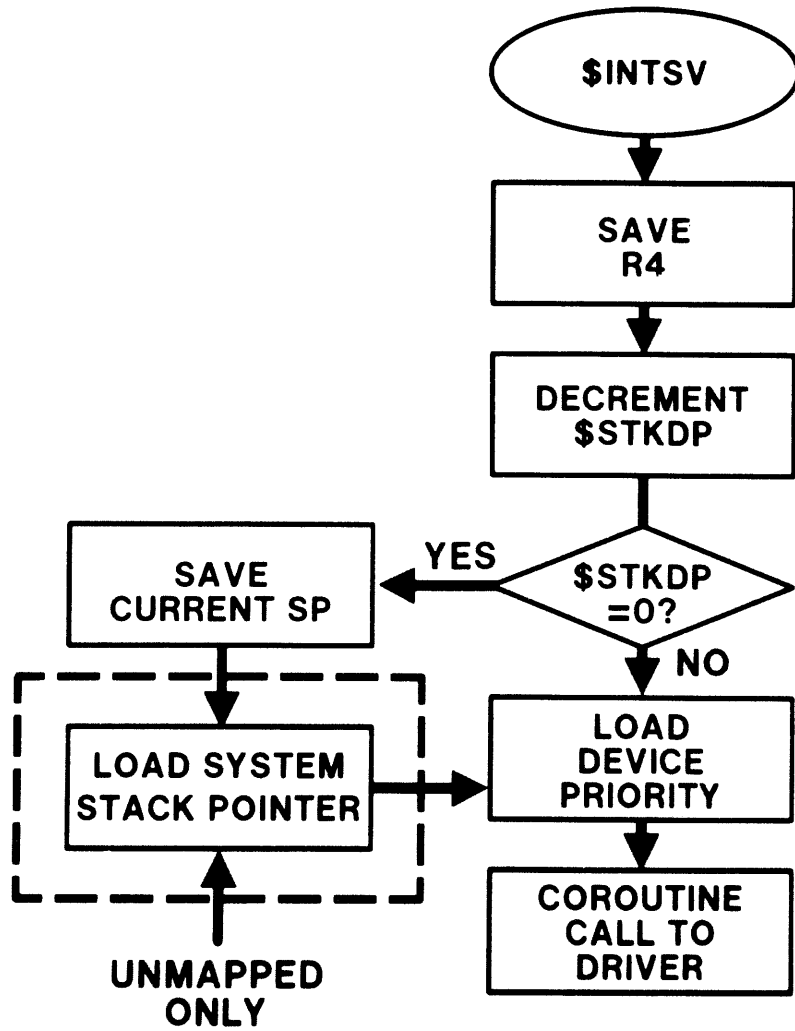
Figure 8-7   The Interrupt Save Routine
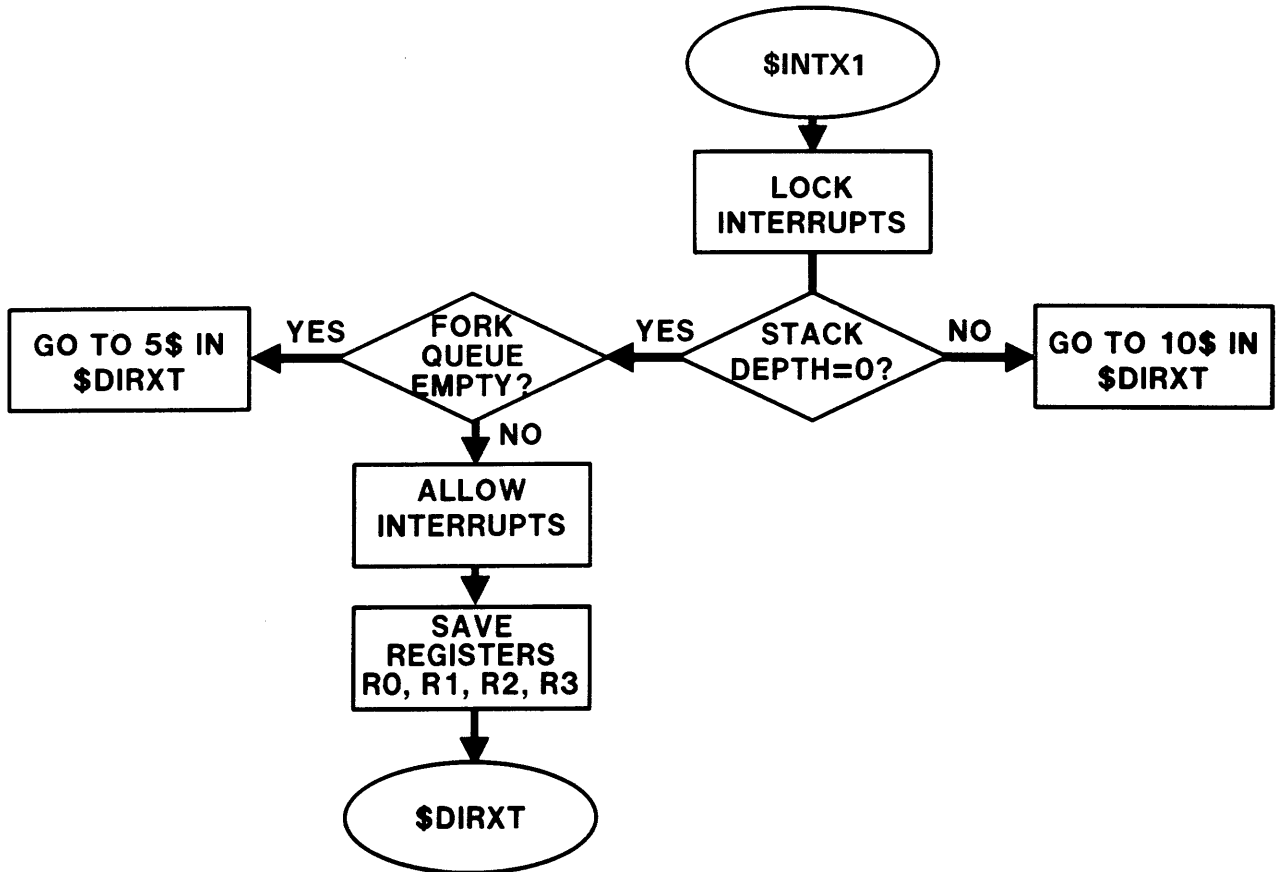
Figure 8-8   The Interrupt Exit Routine

FORK PROCESSING

- Provides serial access to the system data base

    Processes needing access are queued

    No process gains access until the current process is finished with the system data base

- Usually process is part of a device driver, but any privileged task can create a FORK process

- Created by routines in module SYSXT

    $FORK   - Used by device drivers

    $FORK0 - Creates FORK process without saving R4 and R5

    $FORK1 - Creates FORK process saving R4 and R5

    $FORK2 - Used by connect-to-interrupt routines

The FORK Block

- Four word block containing

    Link word for FORK list

    Saved PC

    Saved R5

    Saved R4

- FORK block for device driver contained in the SCB

The FORK List

- FORK blocks are linked into FIFO queue called FORK list
- Begins at list head in SYSCM at location $FRKHD

	S.PKT
POINTER TO NEXT FORK BLOCK	**S.FRK**
SAVED PC	
SAVED R5	
SAVED R4	
KISAR5	
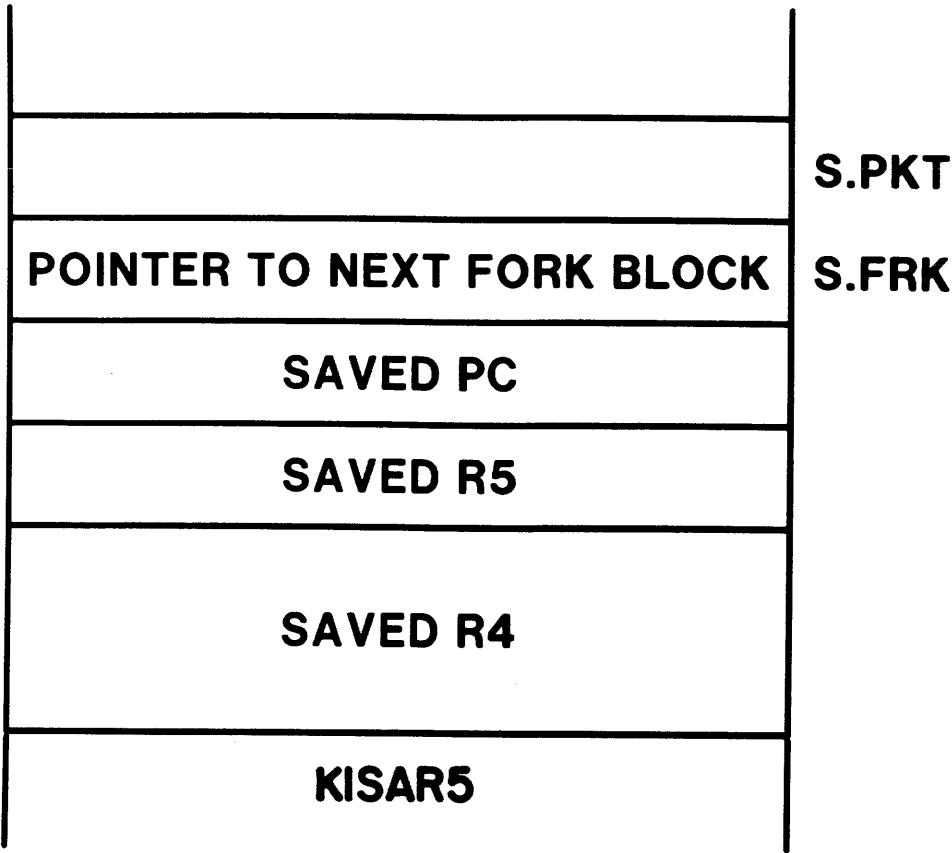
Figure 8-9   The FORK Block in the SCB

SYSCM

$FRKHD
→ FORK BLOCK → FORK BLOCK → FORK BLOCK
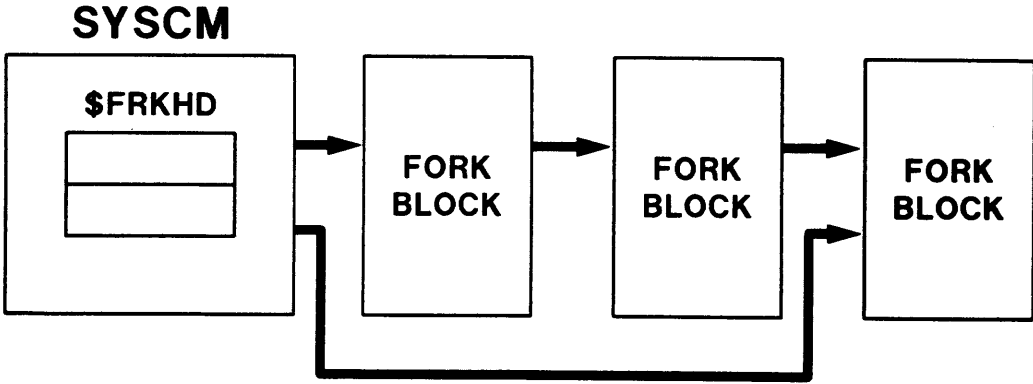
Figure 8-10   The FORK List
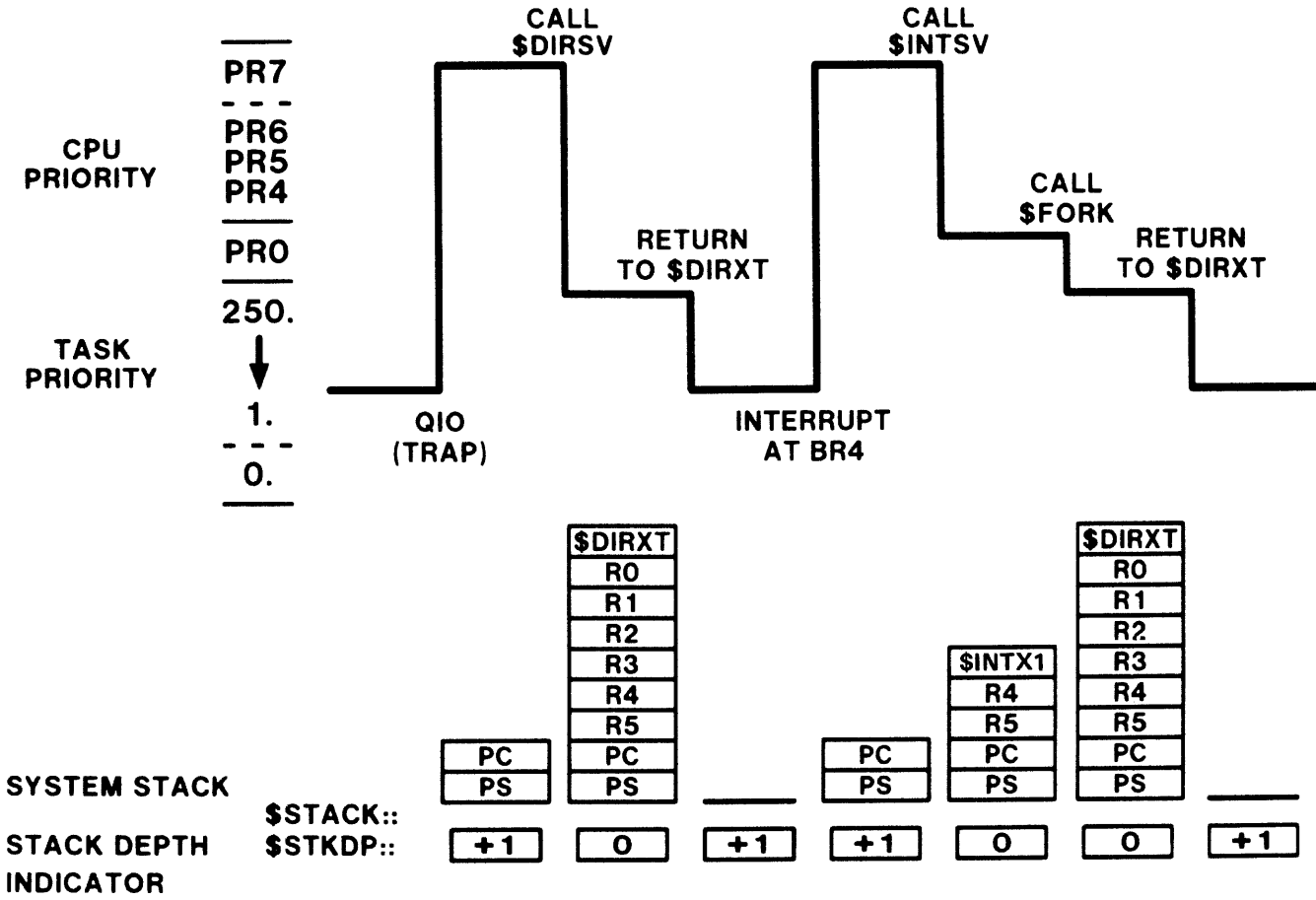
Figure 8-11   Interrupting a Task
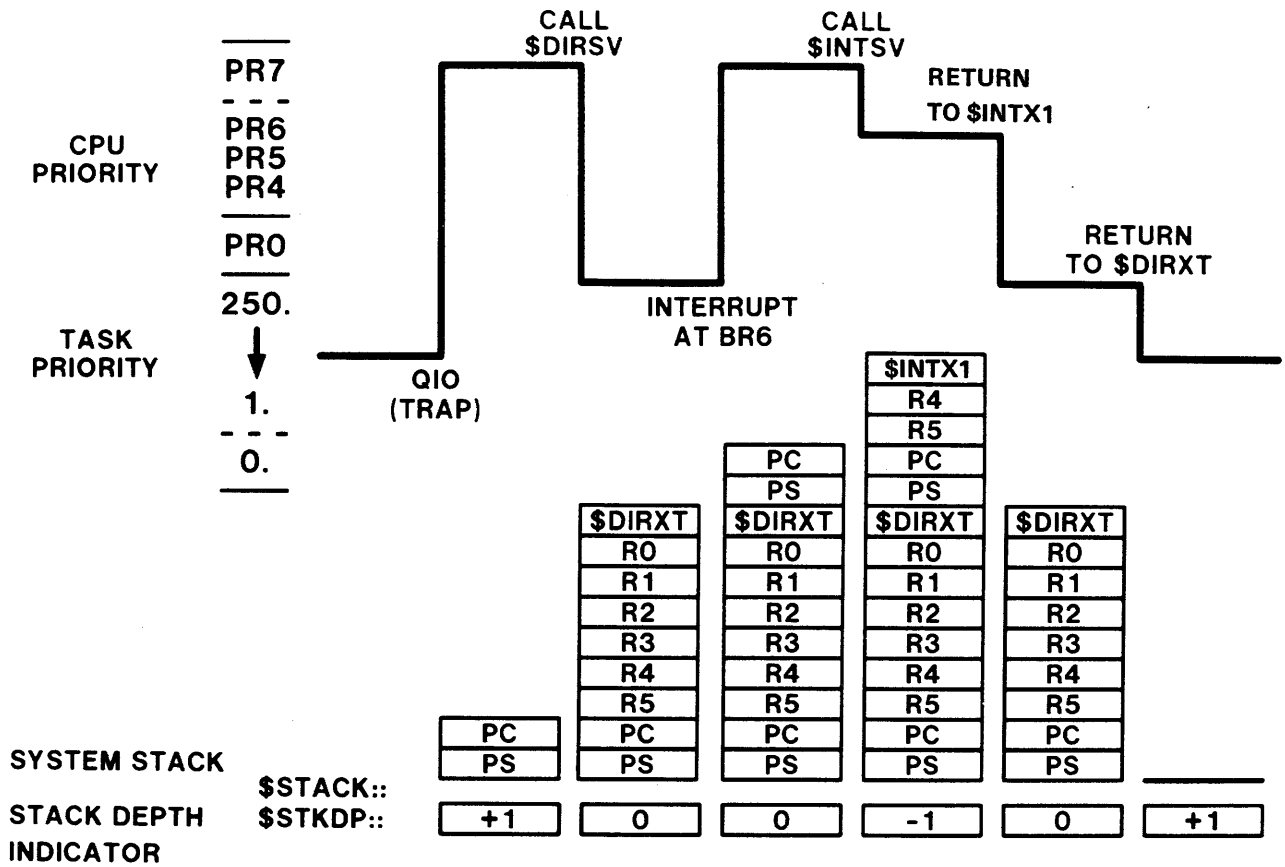
Figure 8-12   Interrupt Executive Processing a Directive
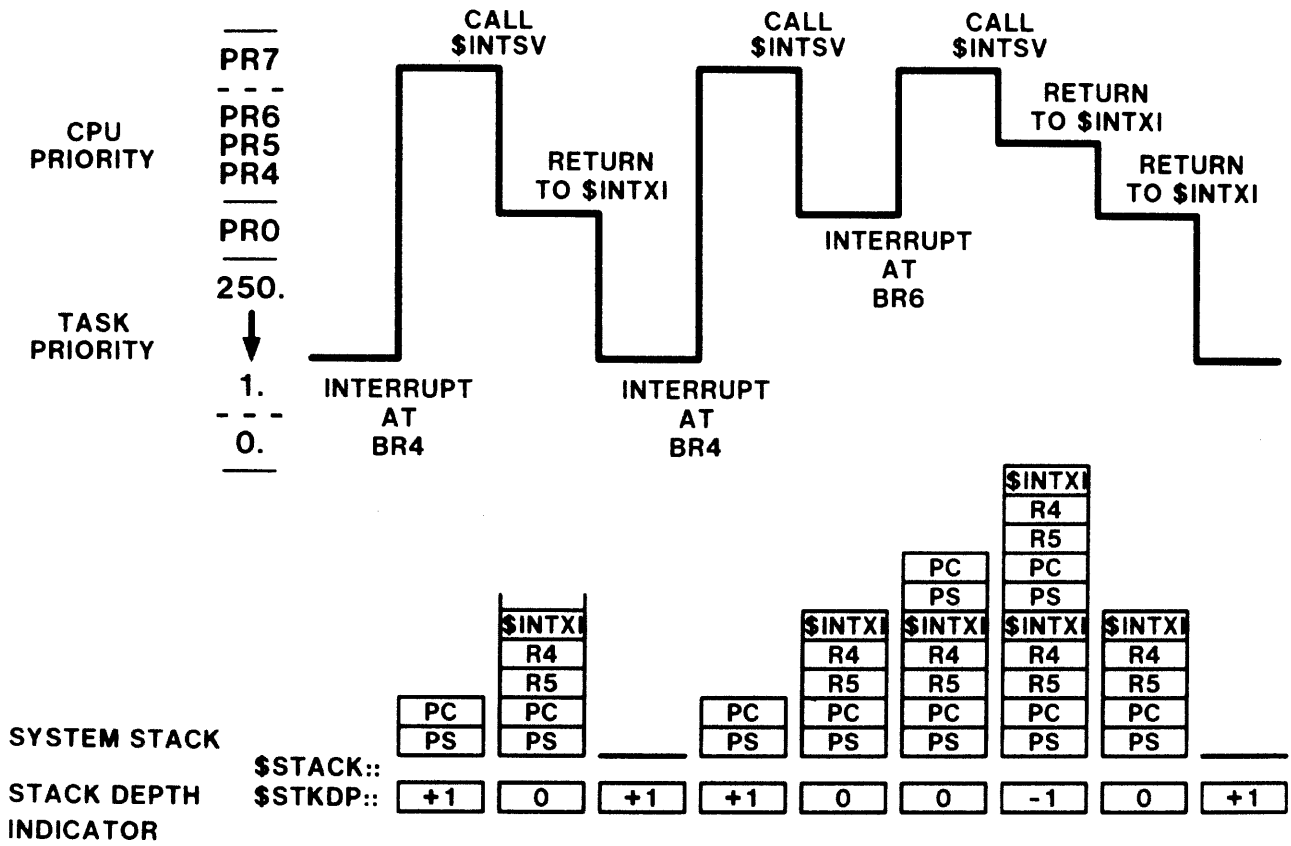
Figure 8-13 Interrupt Executive Processing an Interrupt
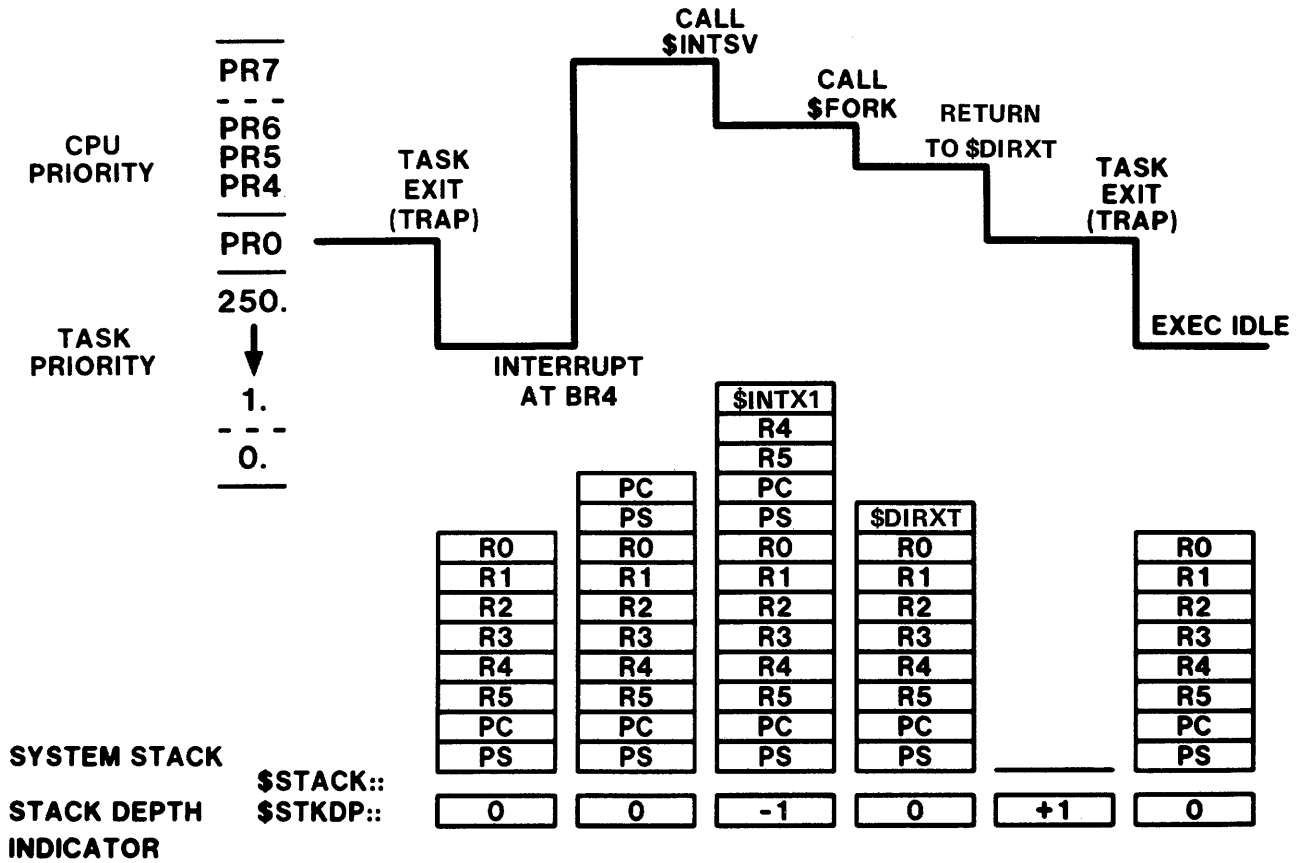
Figure 8-14   Interrupt Executive in an Idle Loop

PRIVILEGED TASKS

- Privilege bit is set in second status word in TCB

    Task can use privileged directives

    Task can perform privileged I/O functions

    Task can perform logical block transfers to/from mounted volumes

- Task is built privileged by using the /PR switch when taskbuilding

    /PR:0 creates privileged task whose mapping is the same as that of a nonprivileged task

    /PR:n creates privileged task which maps to the Executive

        n = 4 if Executive is 16K words
        n = 5 if Executive is 20K words

Processing a Privileged Task

- Privileged task must switch to KERNEL mode to access system data base

- Switch to KERNEL mode is performed by CALL $SWSTK,LABEL instruction

    CALL macro translates this to EMT 376

    CALL macro is in SY:[11,10]RSXMC.MAC

    LABEL is a location in the privileged task to return when leaving KERNEL mode

- In module DRDSP, a jump to location $SWSTK in module SYSXT is executed

- The routine $SWSTK

    Moves the contents of USER mode APRs 5 and 6 to KERNEL mode APRs 5 and 6

    Moves the address of LABEL to location SP+16 on the stack
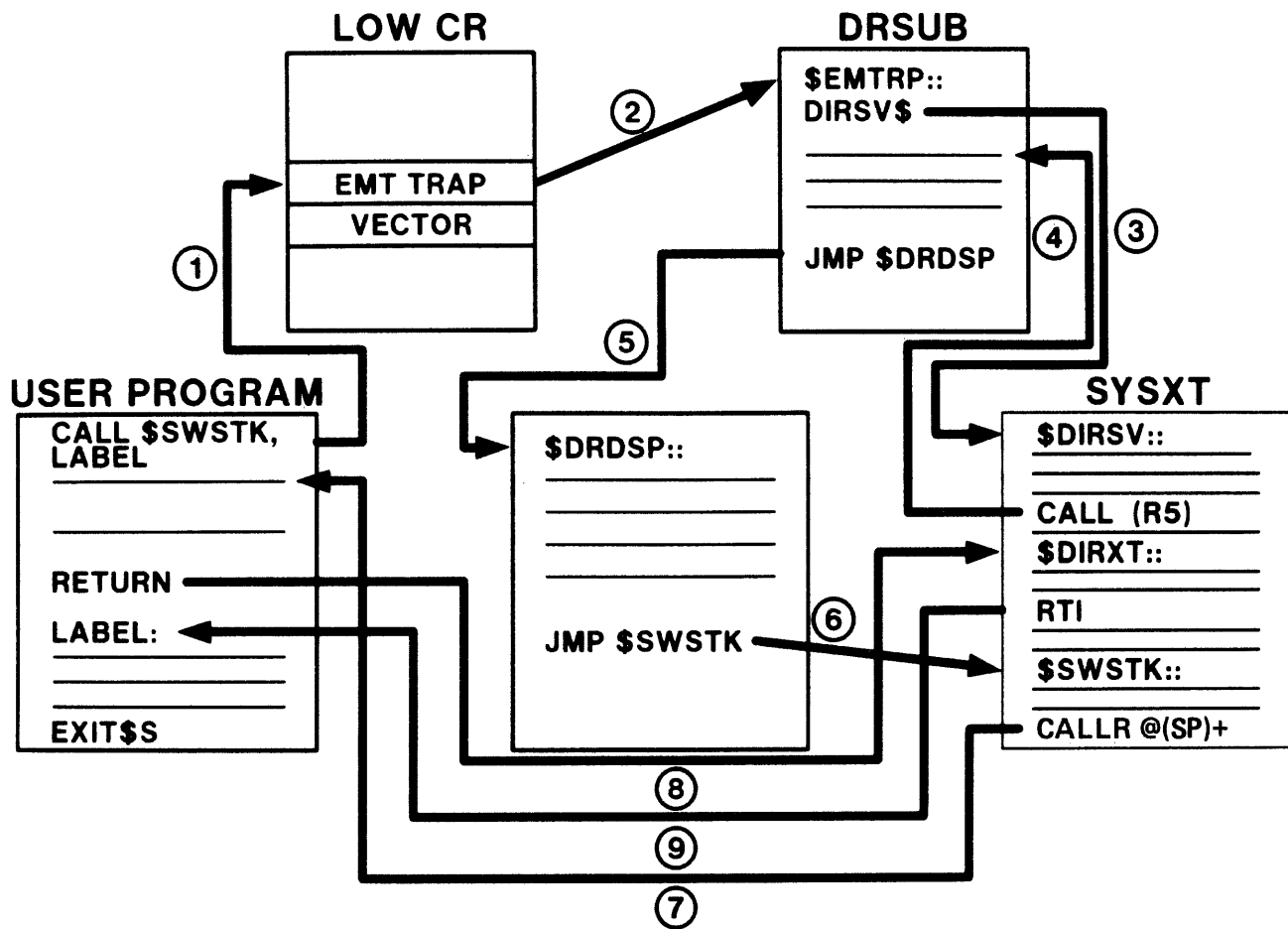
Figure 8-15   Privileged Task Processing
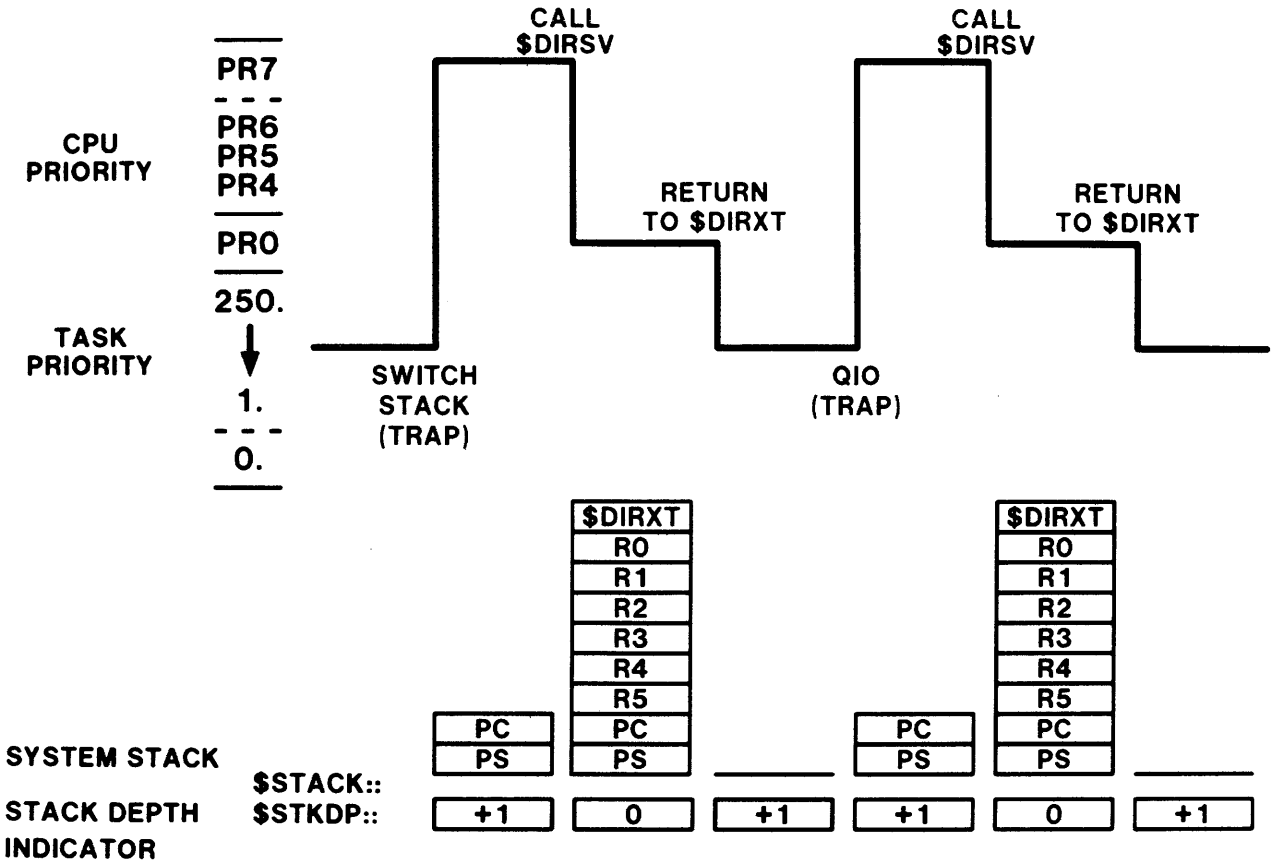
Figure 8-16   Privileged Task Switches Stack and Returns
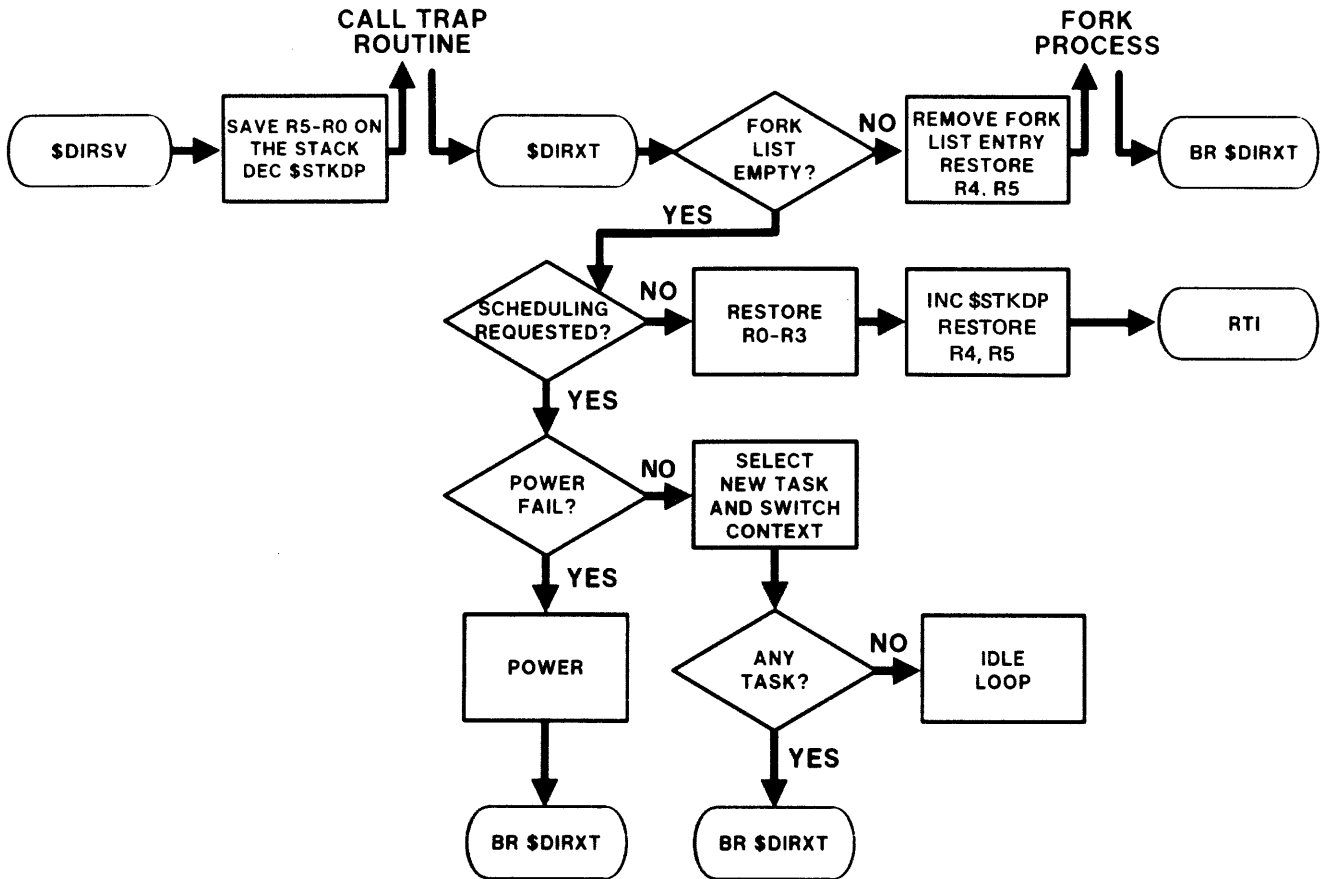
Figure 8-17   System Synchronization Logic: $DIRSV
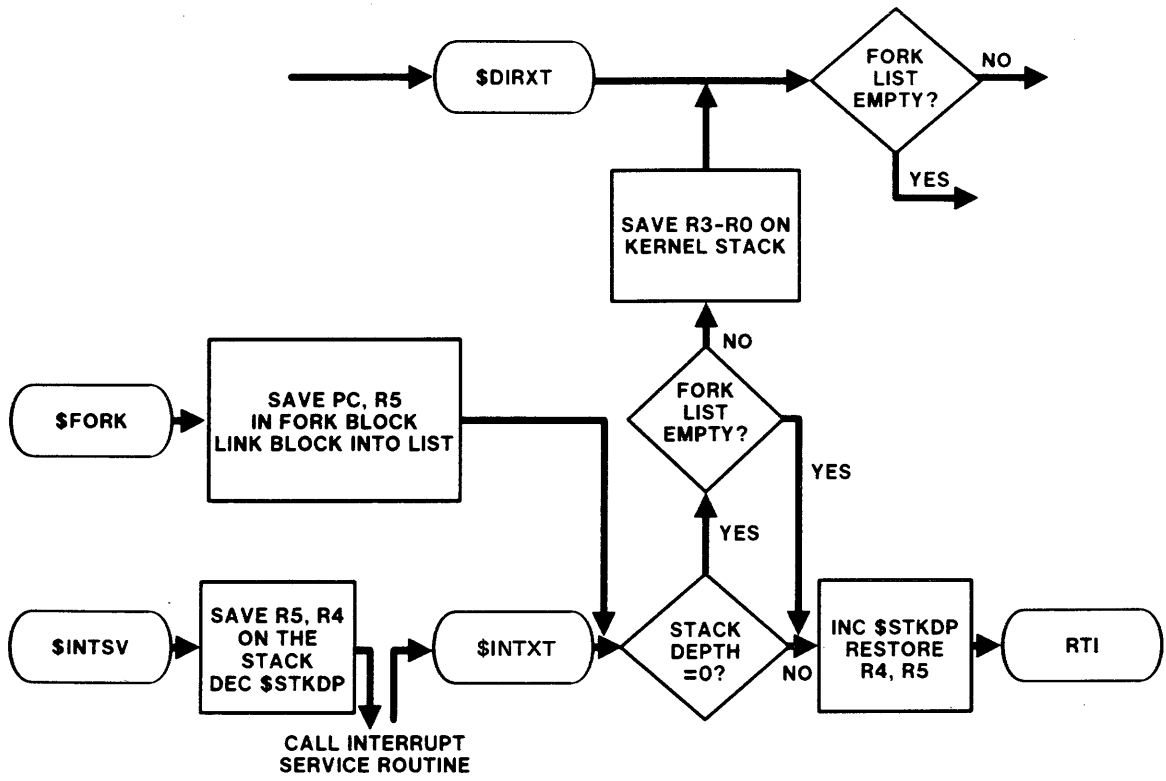
Figure 8-17   System Synchronization Logic: $DIRXT, $FORK,
and $INTSV

WRITING PRIVILEGED TASKS

9

INTRODUCTION

In many practical environments, it is often necessary to write tasks which must interact with the Executive in ways not planned by the designers of the operating system. The tool used in these situations is the privileged task. (In this module, "privileged task" refers to a task which maps to the Executive.) A privileged task has access to both the system data base and Executive code and can therefore act as an extension of the operating system.

To write and use privileged tasks, the programmer must understand the operating system, the system data base and the methods that the Executive uses to synchronize access to the system data base. The previous modules, together with this module, provides that understanding.

OBJECTIVES

1. Assemble and taskbuild privileged tasks.

2. Use a privileged task to access the system data base.

3. Use Executive routines to perform tasks.

RESOURCES

1. RSX-11M System Lists and Data Structures

OVERVIEW

Why Write a Privileged Task

- To acquire the privileged attribute

    Allows task to issue privileged directives

    Allows access to any file without protection checks

    Allows task to issue logical block I/O to a mounted volume

- Tasks which map to the Executive have access to:

    Routines contained in the Executive

    Executive data base including

        System Common (SYSCM)

        System device tables (SYSTB)

        Dynamic Storage Region (POOL)

    The I/O Page

        Device registers

        Memory management registers

Privileged Task Structure

● Consists of at least three sections

    First section

        Executes in USER mode

        Contains the transfer point of the task

        Contains code for processing before transferring to system state

    Second section

        Executes in KERNEL mode

        Contains code which accesses Executive data

    Third section

        Executes in USER mode

        Contains return location from KERNEL mode

        Contains code for processing data obtained from the Executive in the second section

● USER Mode APRs

    For system with 16K word Executive

        APRs Ø - 3 are used to map the Executive

        APRs 4 - 6 are used to map the privileged task

        APR 7 is normally used to map the I/O page but can map an added 4K words of the task if needed

        Privileged task can be

            12K words if I/O page is needed in USER mode

            16K words if I/O page is not needed in USER mode

WRITING PRIVILEGED TASKS

- For system with 20K word Executive

    APRs 0 - 4 are used to map the Executive

    APRs 5 and 6 are used to map the privileged task

    APR 7 is normally used to map the I/O page but can map an added 4K words of the task if needed

    Privileged task can be

    8K words if I/O page is needed in USER mode

    12K words if I/O page is not needed in USER mode

Restrictions on Privileged Tasks

● Parameter Passing

    User state to system state

        Values can be passed in registers R0 - R5

        Values can be passed in memory locations

    System state to user state

        Values can only be passed in memory locations since
        registers are restored

● Time spent in system state should be minimal

    Interrupt processing affected since the FORK queue is not
    serviced

    No other user task can execute

● No trap can occur in system state

    No directives can be executed

    Traps in system state cause a system crash

● Privileged instructions should be used with care

CREATING A PRIVILEGED TASK

Assembling a Privileged Task

- To assemble a privileged task to map to the Executive:

  >MAC File,File=[1,1]EXEMC/ML,[11,10]RSXMC/PA:1,[USER UIC]File

- EXEMC.MLB

  Contains

  Macros defining offsets

  Bit values for all system data structures

  Not needed if no reference is made to these structures

- RSXMC.MAC

  Contains

  Conditional assembly symbols defining the system

  Several commonly used macros such as CALL and CALLR

  Always needed to interpret CALL $SWSTK

Taskbuilding a Privileged Task

- The following command is needed to taskbuild a privileged task which maps to the Executive

  >TKB File/PR:n=File,[1,1]EXELIB/LB,[1,54]RSX11M.STB

- EXELIB.

  Executive object library

  Contains global symbols for resolving offsets in data structures not resolved during assembly

- RSX11M.STB

  Executive symbol table file

  Allows resolution of Executive global symbols used in privileged task

SAMPLE PRIVILEGED TASKS

Tracing Through System Lists

- Function of privileged task

    Calculate and display size of free pool

- The pointer $CRAVL is used to access the list of free pool blocks

- $SWSTK is used to switch to system state

- Time spent in system state is minimal

    System data accessed in system state

- I/O performed in user

    No traps in system state

- Size of free pool is saved before return to user state

    Registers cannot be used to pass data from system to user state

Table 9-1 Pointers to System Lists

Pointer*	System List
$LOGHD	Logical device assignment list
$CRAVL	Free pool block listhead
$ACTHD	Active task list listhead
$CFLPT	Pointer to first checkpoint file PCB
$FRKHD	FORK queue listhead
$CLKHD	Clock queue listhead
$PARHD	Pointer to partition list
$TSKHD	Pointer to system task directory
$GGEF	Pointer to group global event flags list
$ERHEA	Error logging message queue listhead
$PKAVL	Pointer to list of preallocated I/O packets
$UMRHD	Mapping assignment block listhead
$UMRWT	UMR wait queue listhead
$MOULS	Mounted device listhead

*All pointers are located in SYSCM

Table 9-2 Pointers to Data Structures

Pointer*	Description
$HEADR	Pointer to current task header
$TKNPT	Pointer to TKTN TCB
$SHFPT	Pointer to shuffler TCB
$MCRPT	Pointer to MCR TCB
$TKTCB	Pointer to current task TCB
$LDRPT	Pointer to loader TCB

*All pointers are located in SYSCM

```
 1                                        .TITLE  POOLA
 2                              ;+
 3                              ;** - POOLA - DISPLAY POOL SIZE
 4                              ;
 5                              ;       THIS TASK WILL CALCULATE AND DISPLAY THE TOTAL SYSTEM
 6                              ;       DYNAMIC STORAGE REGION SIZE
 7                              ;-
 8                                        .MCALL  QIOW$,EXIT$S,DIR$
 9                                        .NLIST  BEX
10 000000           OUTQIO: QIOW$   IO.WVB,1,1,,,,<OUTBUF,0,40>
11 000030           ARGLST: .BLKW   1               ; ARG LIST FOR $EDMSG
12 000032    124    117   124 OUTMES: .ASCIZ  /TOTAL POOL = %D./
13         000045           BUFFL   =       .-OUTMES+20.
14 000053           OUTBUF: .BLKB   BUFFL
15                                        .EVEN
16 000120           START:
17 000120  005001                         CLR     R1              ; FOR TOTAL FREE SPACE
18 000122  012700  000000G                MOV     #$CRAVL,R0      ; POINT TO FIRST DSR PACKET
19 000126                                 CALL    $SWSTK,30$      ; SWITCH TO SYSTEM STATE
20 000132  066001  000002 10$:            ADD     2(R0),R1        ;; ADD IN SIZE OF THIS PACKET
21 000136  011000                         MOV     (R0),R0         ;; GET LINK TO NEXT PACKET
22 000140  001374                         BNE     10$             ;; IF NE NOT END OF LIST
23 000142  006201                         ASR     R1              ;; CONVERT TO WORDS
24 000144  010167  177660                 MOV     R1,ARGLST       ;; SAVE SIZE OF DSR
25 000150  000207                         RETURN                  ;; RETURN TO USER STATE
26                              ;
27                              ;       FORMAT AND PRINT OUTPUT
28                              ;
29 000152  012700  000053' 30$:           MOV     #OUTBUF,R0      ; SET OUTPUT STRING
30 000156  012701  000032'                MOV     #OUTMES,R1      ; AND INPUT STRING
31 000162  012702  000030'                MOV     #ARGLST,R2      ; AND ARGUMENT LIST
32 000166                                 CALL    $EDMSG          ; FOR EDIT MESSAGE
33 000172  010167  177620                 MOV     R1,OUTQIO+Q.IOPL+2 ; UPDATE MESSAGE SIZE
34 000176                                 DIR$    #OUTQIO         ; WRITE OUT MESSAGE
35 000204                                 EXIT$S                  ; AND EXIT FROM PROGRAM
36         000120'                         .END    START
```

Example 9-1   Tracing Through System Lists
(Sheet 1 of 2)

WRITING PRIVILEGED TASKS

ARGLST 000030R	G$$TTK= 000000	L$$P11= 000002	Q.IOSB= 000010	T$$BTW= 000000
A$$CHK= 000000	H$$RTZ= 000074	L$$11R= 000000	R$$CON= 000000	T$$CCA= 000000
A$$CLI= 000020	IO.WVB= ****** GX	M$$CRB= 000124	R$$DER= 000000	T$$COM= 000000
A$$CNT= 000000	I$$CSZ= 000200	M$$CRX= 000000	R$$DSP= 000000	T$$CON= 000000
A$$CPS= 000000	I$$RAR= 000000	M$$EXT= 000000	R$$EIS= 000000	T$$CTR= 000000
A$$NSI= 000000	I$$RDN= 000000	M$$FCS= 000000	R$$EXV= 000000	T$$CUP= 000000
A$$PRI= 000000	K$$CNT= 177546	M$$MGE= 000000	R$$HDV= 000000	T$$C11= 000001
A$$TRP= 000000	K$$CSR= 177546	M$$MUP= 000000	R$$H11= 000004	T$$ESC= 000000
BUFFL = 000045	K$$DAS= 000000	M$$NET= 002627	R$$JP1= 000004	T$$GMC= 000000
C$$CDA= 000002	K$$IEN= 000115	M$$OVR= 000000	R$$JS1= 000004	T$$GTS= 000000
C$$CKP= 000000	K$$LDC= 000001	M$$XLN= 000400	R$$K11= 000001	T$$KMG= 000000
C$$INT= 000000	K$$TPS= 000074	N$$LDV= 000001	R$$LKL= 000001	T$$LWC= 000000
C$$ORE= 001000	LD$CO = 000000	N$$MOV= 000041	R$$L11= 000001	T$$M03= 000002
C$$RSH= 177564	LD$DB = 000000	N$$UMR= 000030	R$$MPL= 000000	T$$RED= 000000
C$$RUN= 000000	LD$DD = 000000	OUTBUF 000053R	R$$M11= 000004	T$$RNE= 000000
D$$H11= 000003	LD$DK = 000000	OUTMES 000032R	R$$NDC= 000006	T$$RPR= 000000
D$$IAG= 000000	LD$DL = 000000	OUTQIO 000000R	R$$NDH= 000226	T$$RST= 000000
D$$ISK= 000000	LD$DM = 000000	P$$D70= 000000	R$$NDL= 000001	T$$RUB= 000000
D$$L11= 000001	LD$DR = 000000	P$$GMX= 000000	R$$SND= 000000	T$$SMC= 000000
D$$M11= 000003	LD$DS = 000000	P$$LAS= 000000	R$$X21= 000001	T$$UTO= 000036
D$$PAR= 000000	LD$DT = 000000	P$$MAX= 000400	R$$11M= 000000	T$$U58= 000001
D$$SHF= 000000	LD$DY = 000000	P$$OFF= 000000	R$$611= 000001	U$$DAS= 000000
D$$WCK= 000000	LD$LP = 000000	P$$OOL= 000000	START 000120R	U$$MHI= 000000
D$$YNC= 000000	LD$MM = 000000	P$$P45= 000000	S$$ECC= 000000	U$$MLO= 000000
D$$YNM= 000000	LD$NL = 000000	P$$RFL= 000000	S$$HDW= 000000	U$$MRN= 170230
D$$ZMD= 000000	LD$RD = 000000	P$$RTY= 000000	S$$LIB= 000000	V$$CTR= 000774
D$$Z11= 000001	LD$TT = 000000	P$$SRF= 000000	S$$NM1= 042513	V$$TLD= 000120
E$$DVC= 000000	LD$VT = 000000	P$$WND= 000000	S$$NM2= 046522	V$$TLM= 000205
E$$NSI= 000000	LK.SPN= 000002	Q$$OPT= 000005	S$$NM3= 052111	V$$TRM= 000000
E$$PER= 000000	LK.WAT= 000010	Q.IOAE= 000012	S$$OPT= 000000	X$$DBT= 000000
E$$XPR= 000000	L$$ASG= 000000	Q.IOEF= 000006	S$$UBD= 000000	X$$HDR= 000000
F$$LPP= 000000	L$$DBG= 000000	Q.IOFN= 000002	S$$WPC= 000036	$CRAVL= ****** GX
F$$LVL= 000001	L$$DRV= 000000	Q.IOLU= 000004	S$$WPR= 000005	$EDMSG= ****** GX
G$$TPP= 000000	L$$LDR= 000000	Q.IOPL= 000014	S$$WRG= 000000	$$$ARG= 000003
G$$TSS= 000000	L$$PTO= 000170	Q.IOPR= 000007	S$$YSZ= 044000	$$$OST= 000014

```
. ABS.   000000      000
         000212      001
ERRORS DETECTED:   0

VIRTUAL MEMORY USED:   11375 WORDS   ( 45 PAGES)
DYNAMIC MEMORY:   12692 WORDS   ( 48 PAGES)
ELAPSED TIME:   00:00:12
,POOLA=LB:[2,54]RSXMC/PA:1,[1,1]EXEMC/ML,DB2:[305,306]POOLA
```

Example 9-1   Tracing Through System Lists
(Sheet 2 of 2)

Calling Executive Routines

- Function of privileged task

    To simulate issuing an MCR command from a terminal

- Privileged task must construct MCR command block

    Terminal UCB is located by searching device list

    MCR command block (84 bytes) is allocated from pool by using routine $ALOCB

    Block is queued to MCR by using routine $QMCRL

    Block is deallocated on I/O error by calling $DEACB

- Time spent in system state is minimized by returning to user state to perform calculations, etc.

Table 9-3 Executive Pool Routines*

Entry Point	Description	Inputs	Outputs
$ALOCB	Allocates a variable length core buffer First fit Length in multiples of four bytes	R1 = Size of buffer	C-bit = 0 Block allocated C-bit = 1 Insufficient space R0 - Address of block R1 - Length of block
$DEACB	Deallocates a variable length core buffer	R0 = Address of block R1 = Block size	None

*All routines are in Executive module CORAL

238

Table 9-4   Executive Queueing Routines*

Entry Point	Description	Inputs	Outputs
$QINSF	Inserts entry at end of FIFO queue Saves RØ, R1	RØ = Address of listhead R1 = Address of new entry	None
$QINSP	Inserts entry in a priority ordered queue Saves RØ, R1	(Same as $QINSF)	None
$QINSB	Inserts entry at beginning of queue All registers are saved	(Same as $QINSF)	
$QRMVF	Removes entry from beginning of queue Saves RØ	RØ = Address of listhead	C-bit = Ø Entry removed C-bit = 1 Empty list R1 - Address of entry
$QRMVT	Removes entry with a specifed TCB address	RØ = Address of listhead R1 = TCB address	C-bit = Ø Entry removed C-bit = 1 Entry not found R1 - Address of entry
$QMCRL	Queues a command line to CLI of issuing terminal	R1 = Address of command line	None

*All routines are in Executive model QUEUE

Table 9-5   Other Executive Routines

Entry Point	Description	Inputs	Outputs
$SRSTD	Searches STD for specified task Module: REQSB	R3 = Address of RAD50 task name	C-bit = 0 Task found C-bit = 1 Task not found R0 = TCB address of task
$SRNAM	Search the partition list for a named partition Module: PLSUB	R3 = Address of RAD50 name	C-bit = 0 Partition found C-bit = 1 Not found R2 = PCB address
$CEFI	Converts an event flag number to an event flag address Module: EXESB	R0 = Event flag number R5 = TCB address of task	R0 = Mask word R1 = Event flag address

```
      1                                               .TITLE   TERM
      2                                         ;+
      3                                         ;
      4                                         ;   THIS TASK ASKS THE USER FOR A TERMINAL NUMBER AND A
      5                                         ;   COMMAND LINE.  IT THEN PASSES THE COMMAND TO MCR
      6                                         ;   AS IF HAD ORIGINATED AT THE SPECIFIED TERMINAL.  IT
      7                                         ;   PERFORMS THIS TASK BY USING SYSTEM SUBROUTINES, $ALOCB
      8                                         ;   AND $QMCRL, TO ALLOCATE A BLOCK FROM POOL AND PASS IT
      9                                         ;   TO MCR.
     10                                         ;
     11                                         ;   THE TASK MUST BE BUILT PRIVILEGED AND MAP TO THE EXECUTIVE.
     12                                         ;
     13                                         ;   TO ASSEMBLE:
     14                                         ;
     15                                         ;   >MAC TERM=[1,1]EXEMC/ML,[11,10]RSXMC/PA:1,[(YOUR UIC)]TERM
     16                                         ;
     17                                         ;   TO BUILD:
     18                                         ;
     19                                         ;   >TKB TERM/PR:5=TERM,[1,54]RSX11M.STB/SS
     20                                         ;
     21                                         ;+
     22                                               .MCALL   EXIT$S,QIOW$S,DCBDF$
     23 000000                                         DCBDF$
     24                                         ;
     25                                         ;   TERMINAL MESSAGES
     26                                         ;
     27 000000    012   015   105        PRMT:  .ASCII   <12><15>/ENTER A TERMINAL NUMBER: /      ;PROMPT
        000003    116   124   105
        000006    122   040   101
        000011    040   124   105
        000014    122   115   111
        000017    116   101   114
        000022    040   116   125
        000025    115   102   105
        000030    122   072   040
     28            000033                   PLEN   =        .-PRMT                       ;LENGTH OF PROMPT
     29 000033    105   116   124        MES1:  .ASCII   /ENTER A COMMAND FOR MCR/
        000036    105   122   040
        000041    101   040   103
        000044    117   115   115
        000047    101   116   104
        000052    040   106   117
        000055    122   040   115
        000060    103   122
     30            000027                   LENT1  =        .-MES1
     31 000062    116   125   115        MESS1: .ASCII   /NUMBER IS NOT OCTAL, TRY AGAIN!/
        000065    102   105   122
        000070    040   111   123
        000073    040   116   117
        000076    124   040   117
        000101    103   124   101
        000104    114   054   040
        000107    124   122   131
        000112    040   101   107
        000115    101   111   116
        000120    041
     32            000037                   LEN1   =        .-MESS1
```

Example 9-2   Calling Executive Routines
(Sheet 1 of 5)

```
 33 000121    124    105    122  ERMES:  .ASCII  /TERMINAL NOT FOUND!/
    000124    115    111    116
    000127    101    114    040
    000132    116    117    124
    000135    040    106    117
    000140    125    116    104
    000143    041
 34          000023              ERL     =       .-ERMES
 35 000144    116    117    040  POOL:   .ASCII  /NO POOL AVAILABLE!/
    000147    120    117    117
    000152    114    040    101
    000155    126    101    111
    000160    114    101    102
    000163    114    105    041
 36          000022              PL      =       .-POOL
 37 000166    111    057    117  IOERR:  .ASCII  !I/O ERROR ON COMMAND INPUT!
    000171    040    105    122
    000174    122    117    122
    000177    040    117    116
    000202    040    103    117
    000205    115    115    101
    000210    116    104    040
    000213    111    116    120
    000216    125    124
 38          000032              IOL     =       .-IOERR
 39                                      .EVEN                        ;MAKE SURE OF WORD BOUNDARY
 40                              ;
 41                              ;       PROGRAM DATA
 42                              ;
 43 000220                      BUF:    .BLKB   4                    ;BUFFER FOR TERMINAL NUMBER
 44 000224                      IOSB:   .BLKW   2                    ;I/O STATUS BLOCK
 45 000230    124    124        NAM:    .ASCII  /TT/                 ;TERMINAL NAME
 46 000232    177777            ERRCNT: .WORD   -1                   ;ERROR INDICATOR
 47 000234    000000            COMM:   .WORD   0                    ;ADDRESS OF COMMAND LINE IN POOL
 48 000236    000000            PLACE:  .WORD   0                    ;ADDRESS OF POOL BLOCK
 49 000240    000000            UCB:    .WORD   0                    ;SAVED UCB ADDRESS
 50 000242    000000            DCB:    .WORD   0                    ;SAVED DCB ADDRESS
 51 000244    000000            LOW:    .WORD   0                    ;SAVED LOW UNIT NUMBER
 52 000246    000000            HIGH:   .WORD   0                    ;SAVED HIGH UNIT NUMBER
 53                              ;
 54                              ;       BEGINNING OF CODE
 55                              ;
 56 000250                      TERM:   QIOW$S  #IO.WVB,#5,#1,,,,<#PRMT,#PLEN,#40>        ;PROMPT
 57 000322                              QIOW$S  #IO.RVB,#5,#1,,#IOSB,,<#BUF,#4> ;GET 'TT' NUMBER
 58 000374    016703 177626             MOV     IOSB+2,R3            ;SAVE CHARACTER COUNT
 59 000400    012700 000220'            MOV     #BUF,R0              ;SET UP FOR CALL TO CONVERSION
 60 000404    005002                    CLR     R2                   ; ROUTINE - FROM ASCII TO
 61 000406                              CALL    $COTB                ;  BINARY(OCTAL)
 62                                                                  ; R1 WILL CONTAIN CONVERTED NUMBER
 63                                                                  ; R2 WILL CONTAIN TERMINATOR
 64 000412    005702                    TST     R2                   ;TEST FOR ERROR
 65 000414    001402                    BEQ     CONT                 ;IF NOT CONTINUE
 66 000416    000167 000436             JMP     ERR1                 ;GO TO ERROR ROUTINE
 67 000422                      CONT:   CALL    $SWSTK,BACK          ;;SWITCH TO SYSTEM STATE
 68 000426    012700 000000G            MOV     #$DEVHD,R0           ;;PICK UP LISTHEAD FOR DCB LIST
 69 000432    011000            L1:     MOV     (R0),R0              ;;GO TO NEXT DCB
 70 000434    001422                    BEQ     GO                   ;;IF NONE FINISH UP
```

Example 9-2 Calling Executive Routines
(Sheet 2 of 5)

```
71 000436  026760  177566  000004          CMP     NAM,D.NAM(RO)       ;;LOOK FOR 'TT' DCB
72 000444  001372                           BNE     L1                  ;;IF NOT 'TT' GET NEXT DCB
73 000446  116067  000006  177570           MOVB    D.UNIT(RO),LOW      ;;GET LOW UNIT NUMBER
74 000454  116067  000007  177564           MOVB    D.UNIT+1(RO),HIGH   ;;GET HIGH UNIT NUMBER
75 000462  020167  177560                    CMP     R1,HIGH             ;;COMPARE INPUT WITH HIGH
76 000466  003361                           BGT     L1                  ;;IF INPUT LARGER GET NEXT DCB
77 000470  010067  177546                    MOV     RO,DCB              ;;SAVE DCB ADDRESS
78 000474  062767  000001  177530           ADD     #1,ERRCNT           ;;INDICATE NO ERROR HERE
79 000502                           GO:     RETURN                      ;;RETURN TO USER STATE
80 000504                           BACK:                               ;NOW IN USER STATE
81 000504  005767  177522                    TST     ERRCNT              ;TEST FOR ERROR
82 000510  001402                           BEQ     1$                  ;CONTINUE IF OK
83 000512  000167  000430                    JMP     ERR2                ;GO TO ERROR ROUTINE
84 000516  016700  177520          1$:       MOV     DCB,RO              ;GET DCB ADDRESS
85 000522  016002  000002                    MOV     D.UCB(RO),R2        ;GET ADDRESS OF FIRST UCB
86 000526  010104                           MOV     R1,R4               ;MOVE FOR CALCULATIONS
87 000530  166704  177510                    SUB     LOW,R4              ;FIND THE NUMBER WITHIN UCBS
88                                                                       ;  FOR CURRENT DCB
89 000534  016005  000010                    MOV     D.UCBL(RO),R5       ;GET UCB LENGTH
90 000540  070504                           MUL     R4,R5               ;CALCULATE OFFSET FOR UCB
91 000542  060502                           ADD     R5,R2               ;CALCULATE BEGINNING OF UCB
92 000544  010267  177470                    MOV     R2,UCB              ;SAVE UCB ADDRESS
93 000550  012767  177777  177454           MOV     #-1,ERRCNT          ;RESET ERROR COUNT
94 000556                           CALL    $SWSTK,BACK2                ;;SWITCH TO SYSTEM STATE
95 000562  012701  000124                    MOV     #84.,R1             ;;SET BLOCK SIZE
96 000566                           CALL    $ALOCB                      ;;ALLOCATE BLOCK FROM POOL
97 000572  103404                           BCS     NONE                ;;NO ROOM IN POOL
98 000574  005267  177432                    INC     ERRCNT              ;;INDICATE NO ERROR HERE
99 000600  010067  177432                    MOV     RO,PLACE            ;;SAVE ADDRESS OF POOL BLOCK
100 000604                          NONE:    RETURN                     ;;RETURN TO USER STATE
101 000606                          BACK2:                              ;BACK IN USER STATE
102 000606  005767  177420                   TST     ERRCNT             ;CHECK FOR ERROR
103 000612  001402                          BEQ     1$                  ;CONTINUE IF OK
104 000614  000167  000406                   JMP     ERR3               ;GO TO ERROR ROUTINE
105 000620  012700  000124          1$:      MOV     #84.,RO            ;SET LENGTH OF BLOCK
106 000624  016701  177406                   MOV     PLACE,R1           ;GET ADDRESS OF BLOCK
107 000630  112721  000040          2$:      MOVB    #40,(R1)+          ;PLACE BLANKS IN BLOCK
108 000634  077003                          SOB     RO,2$               ;LOOP UNTIL THROUGH
109 000636  016767  177374  177370           MOV     PLACE,COMM         ;CALCULATE LOCATION OF COMMAND
110 000644  062767  000004  177362           ADD     #4,COMM            ;  LOCATION IN BLOCK
111 000652                          QIOW$S   #IO.WVB,#5,#1,,,,<#MES1,#LENT1,#40>     ;PROMPT FOR COMMAND
112 000724  005067  177274                   CLR     IOSB
113 000730  005067  177272                   CLR     IOSB+2
114 000734                          QIOW$S   #IO.RVB,#5,#1,,IOSB,,<COMM,#80.>        ;INPUT COMMAND
115 001006  103002                          BCC     3$                  ;CONTINUE IF OK
116 001010  000167  000272                   JMP     ERR4               ;BRANCH IF QIO ERROR
117 001014  105767  177204          3$:      TSTB    IOSB               ;CHECK FOR I/O ERROR
118 001020  002002                          BGE     4$                  ;CONTINUE IF OK
119 001022  000167  000260                   JMP     ERR4               ;BRANCH IF ERROR
120 001026                          4$:      CALL    $SWSTK,BACK3        ;;SWITCH TO SYSTEM STATE
121 001032  016701  177200                   MOV     PLACE,R1           ;;GET ADDRESS OF BLOCK
122 001036  016761  177176  000002           MOV     UCB,2(R1)          ;;PLACE UCB ADDRESS IN BLOCK
123 001044                          CALL    $QMCRL                      ;;QUEUE BLOCK TO MCR
124 001050                          RETURN                              ;;LEAVE SYSTEM STATE
125 001052                          BACK3:                              ;BACK IN USER STATE
126 001052                          EXIT$S                              ;EXIT
127                                  ;
```

Example 9-2   Calling Executive Routines
(Sheet 3 of 5)

```
128                               ;       ROUTINE TO HANDLE BAD TERMINAL NUMBER
129                               ;
130 001060               ERR1:    QIOW$S  #IO.WVB,#5,#1,,,,<#MESS1,#LEN1,#40>      ;PRINT MESSAGE
131 001132  012700  000220'       MOV     #BUF,RO                     ;GET INPUT BUFFER ADDRESS
132 001136  005020               CLEAR:   CLR     (RO)+                       ;CLEAR BUFFER
133 001140  077302                        SOB     R3,CLEAR                    ;R3 CONTAINS CHARACTER COUNT
134 001142  000167  177102                JMP     TERM                        ;TRY AGAIN
135                               ;
136                               ;       ROUTINE TO HANDLE NON-EXISTENT TERMINAL
137                               ;
138 001146               ERR2:    QIOW$S  #IO.WVB,#5,#1,,,,<#ERMES,#ERL,#40>      ;PRINT MESSAGE
139 001220                        EXIT$S                              ;EXIT
140                               ;
141                               ;       ROUTINE TO HANDLE POOL ALLOCATION FAILURE
142                               ;
143 001226               ERR3:    QIOW$S  #IO.WVB,#5,#1,,,,<#POOL,#PL,#40>      ;PRINT MESSAGE
144 001300                        EXIT$S                              ;EXIT
145                               ;
146                               ;       ROUTINE TO HANDLE I/O ERRORS ON COMMAND INPUT
147                               ;       THE POOL BLOCK MUST BE DEALLOCATED!
148                               ;
149 001306               ERR4:    QIOW$S  #IO.WVB,#5,#1,,,,<#IOERR,#IOL,#40>      ;PRINT MESSAGE
150 001360                        CALL    $SWSTK,FIN                  ;;SWITCH TO SYSTEM STATE
151 001364  016700  176646        MOV     PLACE,RO                    ;;GET ADDRESS OF BLOCK
152 001370  012701  000124        MOV     #84.,R1                     ;;SET BLOCK LENGTH
153 001374                        CALL    $DEACB                      ;;DEALLOCATE BLOCK
154 001400                        RETURN                              ;;LEAVE SYSTEM STATE
155 001402               FIN:                                         ;BACK IN USER STATE
156 001402                        EXIT$S                              ;EXIT
157         000250'               .END    TERM
```

Example 9-2 Calling Executive Routines
(Sheet 4 of 5)

```
A$$BIO= 000000     D.MSK   000014     K$$IEN= 000113     PRMT    000000R     T$$BTW= 000000
A$$BRT= 000000     D.NAM   000004     K$$LDC= 000144     P$$BPR= 000063     T$$BUF= 000000
A$$CHK= 000000     D.PCB   000034     K$$TPS= 000144     P$$CTL= 000000     T$$CCA= 000000
A$$CLI= 000004     D.UCB   000002     LD$CO = 000000     P$$FRS= 000310     T$$CCO= 000000
A$$CPS= 000000     D.UCBL  000010     LD$DD = 000000     P$$GMX= 000000     T$$CPW= 000000
A$$NSI= 000000     D.UNIT  000006     LD$DK = 000000     P$$HIL= 003100     T$$CTR= 000000
A$$PRI= 000000     D.VCAN= 000002     LD$DL = 000000     P$$LAS= 000000     T$$CUP= 000000
A$$TRP= 000000     D.VDEB= 177776     LD$DM = 000000     P$$LOL= 001130     T$$C11= 000001
BACK    000504R    D.VINI= 000000     LD$DR = 000000     P$$OFF= 000000     T$$ESC= 000000
BACK2   000606R    D.VOUT= 000004     LD$DS = 000000     P$$P45= 000000     T$$GMC= 000000
BACK3   001052R    D.VPWF= 000006     LD$DT = 000000     P$$RFL= 000000     T$$GTS= 000000
BUF     000220R    ERL   = 000023     LD$LP = 000000     P$$RTY= 000000     T$$HFF= 000000
B$$LV1= 030463     ERMES   000121R    LD$MM = 000000     P$$SRF= 000000     T$$HLD= 000000
B$$LV2= 020040     ERRCNT  000232R    LD$TT = 000000     Q$$OPT= 000005     T$$J16= 000001
CLEAR   001136R    ERR1    001060R    LENT1 = 000027     R$$DER= 000000     T$$KMG= 000000
COMM    000234R    ERR2    001146R    LEN1  = 000037     R$$EXV= 000000     T$$LWC= 000000
CONT    000422R    ERR3    001226R    LOW     000244R    R$$JS1= 000001     T$$RED= 000000
C$$CDA= 000002     ERR4    001306R    L$$ASG= 000000     R$$K11= 000001     T$$RNE= 000000
C$$CHE= 000000     E$$ACT= 000000     L$$DRV= 000000     R$$LKL= 000001     T$$RPR= 000000
C$$CKP= 000004     E$$DVC= 000000     L$$LDR= 000000     R$$L11= 000001     T$$RST= 000000
C$$CSR= 177404     E$$LOG= 000000     L$$PTO= 000036     R$$MOF= 000000     T$$RUB= 000000
C$$INT= 000000     E$$XPR= 000000     L$$P11= 000001     R$$M11= 000001     T$$SMC= 000000
C$$ONS= 000001     FIN     001402R    L1      000432R    R$$NDC= 000005     T$$SYN= 000000
C$$ORE= 002022     F$$LPP= 000000     MESS1   000062R    R$$NDH= 000226     T$$TRW= 000000
C$$OTM= 000000     F$$LVL= 000001     MES1    000033R    R$$NDL= 000001     T$$UTB= 000000
C$$RSH= 177564     GO      000502R    M$$CRB= 000124     R$$POI= 000000     T$$UTO= 000170
C$$RUN= 000000     G$$EFN= 000000     M$$CRX= 000000     R$$SND= 000000     T$$U58= 000001
C$$SMT= 000000     G$$TPP= 000000     M$$EIS= 000000     R$$11M= 000000     T$$VBF= 000000
C$$TTY= 177564     G$$TSS= 000000     M$$EXT= 000000     R$$60F= 000000     T$$30P= 000000
DCB     000242R    G$$TTK= 000000     M$$FCS= 000000     R$$611= 000001     UCB     000240R
D$$H11= 000001     HIGH    000246R    M$$MGE= 000000     S$$ECC= 000000     U$$MHI= 000000
D$$IAG= 000000     H$$RTZ= 000074     M$$MUP= 000000     S$$NM1= 051522     U$$MLO= 160000
D$$ISK= 000000     IOERR   000166R    M$$OVR= 000000     S$$NM2= 030530     U$$MRN= 170234
D$$L11= 000002     IOL   = 000032     NAM     000230R    S$$NM3= 046461     V$$CTR= 000400
D$$M11= 000001     IOSB    000224R    NONE    000604R    S$$TIM= 000000     V$$RSN= 000040
D$$PAR= 000000     IO.RVB= ****** GX  N$$LDV= 000001     S$$TOP= 000000     X$$DBT= 000000
D$$SHF= 000000     IO.WVB= ****** GX  N$$MOV= 000041     S$$WLK= 000000     $ALOCB= ****** GX
D$$WCK= 000000     I$$RAR= 000000     N$$UMR= 000034     S$$WPC= 000036     $COTB = ****** GX
D$$YNC= 000000     I$$RDN= 000000     PL    = 000022     S$$WPR= 000005     $DEACB= ****** GX
D$$YNM= 000000     K$$AST= 000000     PLACE   000236R    S$$YSZ= 022000     $DEVHD= ****** GX
D.DSP   000012     K$$CNT= 172542     PLEN  = 000033     TERM    000250R    $QMCRL= ****** GX
D.LNK   000000     K$$CSR= 172540     POOL    000144R    T$$ACR= 000000     $$$ARG= 000002
```

```
. ABS.  000036     000
        001410     001
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  10113 WORDS  ( 40 PAGES)
DYNAMIC MEMORY:  16120 WORDS  ( 62 PAGES)
ELAPSED TIME:  00:00:14
TERM,TERM/-SP=[1,1]EXEMC/ML,[11,10]RSXMC/PA:1,[7,1]TERM
```

Example 9-2   Calling Executive Routines
(Sheet 5 of 5)

SYSTEM CRASHES

10

INTRODUCTION

Only certain carefully defined SSTs are allowed in system state. System crashes occur when a software fault is recognized within system state. The processor trap mechanism signals the system crash. CDA is a SYSGEN option which aids the system programmer in tracing system faults. When a fault occurs, the Executive trap handling facility saves the register context in memory, then passes control to the crash routine, which dumps the memory to a disk or tape. The CDA utility may be used to format the dump information into a convenient report.

OBJECTIVES

1. To analyze a report generated by CDA to determine the cause of a system crash.

RESOURCES

1. RSX-11M/M-PLUS Crash Dump Analyzer

2. RSX-11M System Lists and Data Structures

CRASH PROCESSING

Causes of Crashes

- Processor traps in system state

    Powerfail and parity error traps do not cause crash

    Other processor traps cause system crash

    System automatically crashes

- Processor HALTs or enters infinite loop

    System must be manually crashed

- Manually crashing the system

    Physical location 40 contains the instruction JMP $CRASH

        Processor halted

        Address 40 loaded

        Processor restarted

Crash Sequences

- Three different crash sequences

    Initiated by IOT

    Initiated by EMT or TRAP

    Initiated by other trap

        Odd address

        Illegal instruction

        Memory management fault

        Floating-point fault/PIRQ

Crash Sequences ·

- IOT

    Used when system detects an inconsistent system state

    Sequence:

    > IOT trap vectors to the module SSTSR

    > R0 - R5 are not saved

    > Only PS and PC are pushed onto the stack

    > Jump to $CRASH in module CRASH is executed

    Executive issues IOT only in modules DRDSP and SSTSR

- EMT or TRAP

    Results from EMT or TRAP attemp in system state

    Sequence:

    > Directive save operation is performed

    > Executive issues IOT

    > Continue as with the IOT

- Other Traps

    Traps vector to module SSTSR

    Directive save operation is performed

    If Memory Management Fault, the status registers SR0, SR1, and SR2 are pushed onto the stack

    One word fault code is pushed onto the stack

    Byte count of the information saved is pushed onto the stack

    Executive jumps to $CRASH

Obtaining a Crash Dump Analysis

- If XDT is present, respond with X to continue with crash

- Registers are printed on the console printer; the system halts

- Mount crash dump volume and press the continue switch

- After memory contents is saved, boot the system

- Run CDA to produce the analysis

    Inputs: The copy of memory created at the crash or by a previous execution of CDA

    The symbol definition file for the system

    Outputs: A listing file containing the analysis of the dump

    A binary file which is a copy of the contents of the crash dump volume

- Command Format:

    ```
    >CDA <CR>
    CDA>[listfile/sw],[binaryfile/sw]=symbolfile/STB,
    crash-input[/sw]
    ```

- Two types of switches can be used

    Analysis switches - determine which CDA routines are applied to the input

    Function switches - provide control options

CRASH DUMP ANALYSIS

Finding the Cause of a System Crash

- Usually requires

    Output from the CDA utility

    Map of the operating system

    Listings and maps of relevant tasks

- Common steps in interpreting a crash dump

    Get KERNEL stack pointer

    Examine KERNEL stack and $STKDP for sequence of events (i.e., traps and interrupts)

    Find PC and PS when trap occurred (from KERNEL stack)

    Determine APRs in use and find which APR mapped the instruction at which the crash occurred

    Use the Memory Map to locate the proper partition or region

    Use the listing and map of the code occupying the region to locate the instruction

- System data base may have been corrupted long before crash occurred

    'Current Task' may not have caused crash

Table 10-1   CDA Switches

CDA Analysis Switches

/ACT   Lists the TCBs for active tasks

/CLQ   Lists the contents of the clock queue

/DEV   Lists the information on all devices in the system

/HDR   Lists the headers for all resident tasks

/PCB   Lists the PCBs

/POOL   Lists the contents of POOL

/STD   Lists the contents of the STD

/ALL   Lists the output of all analysis routines

/DUMP:a:b
         Lists the contents of memory between the 18-bit
         addresses a and b

/TASK=name:a:b
         Lists the virtual space of task 'name' between the
         16-bit address a and b

Note: All analysis switches are used with the crash-input

CDA Function Switches

/LIMIT:n   Limits output listing to n pages

           Used on the list file

/MEMSIZ:n Saves nK words of memory in a binary file

           Used on the binary file

/STB   Identifies the symbol definition file

           Used on the STB file for the system

Basic Information Provided in a CDA

The KERNEL Stack and $STKDP

- The KERNEL stack provides a record of the system entrances since system state was entered

- Each trap and interrupt causes task context to be saved on the KERNEL stack

- $STKDP indicates how many times context has been saved

- Contents of stack can be partitioned by locating the saved PSs

    Each PS indicates the start of context saved by a trap or an interrupt

    PS on stack located by position on stack

        First entry on stack is PS from first trap or interrupt

        Next PS will be nine words further on stack if trap or fork occurred

        Next PS will be five words further on stack if interrupt occurred

        Context pushed because SST contains 11 words plus extra trap specific words

    If an SST occurred, it is always the last context on stack

Contents of the APRs

- Determines mapping between virtual and physical addresses

    KERNEL APRs show mapping when crash occurred

    USER APRs show mapping of current user task
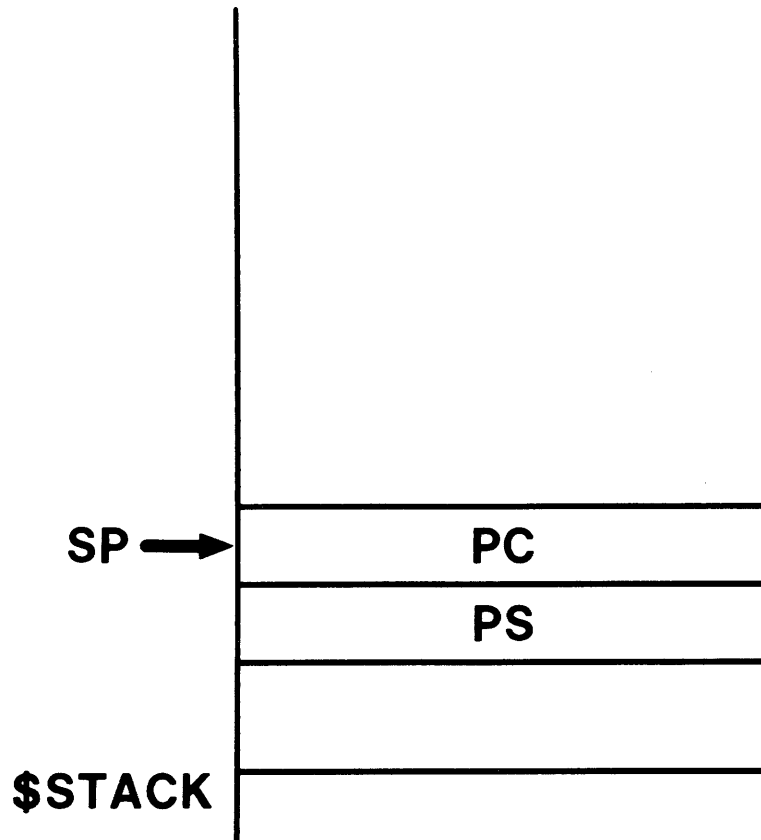
Figure 10-1 The Stack After a System Crash - IOT

SP →	PC
	PS
	RETURN TO $DIRXT
	R0
	R1
	R2
	R3
	R4
	R5
	PC
	PS
$STACK	

Figure 1Ø-2 The Stack After a System Crash - EMT

```
              ┌─────────────────────────┐
              │                         │
    SP ──────▶├─────────────────────────┤
              │       BYTE COUNT         │
              ├─────────────────────────┤
              │       FAULT CODE         │
              ├─────────────────────────┤
              │                         │
              │     TRAP SPECIFIC        │
              │        DATA              │
              ├─────────────────────────┤
              │    RETURN TO $DIRXT      │
              ├─────────────────────────┤
              │          R0              │
              ├─────────────────────────┤
              │          R1              │
              ├─────────────────────────┤
              │          R2              │
              ├─────────────────────────┤
              │          R3              │
              ├─────────────────────────┤
              │          R4              │
              ├─────────────────────────┤
              │          R5              │
              ├─────────────────────────┤
              │          PC              │
              ├─────────────────────────┤
              │          PS              │
              ├─────────────────────────┤
              │                         │
 $STACK       │                         │
              └─────────────────────────┘
```
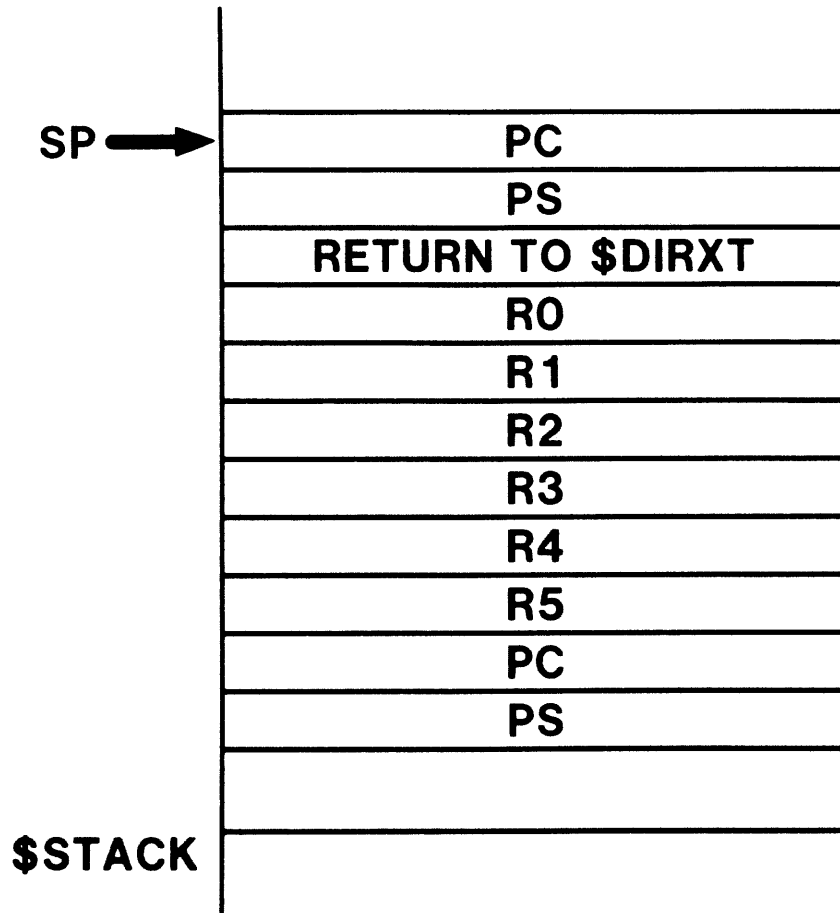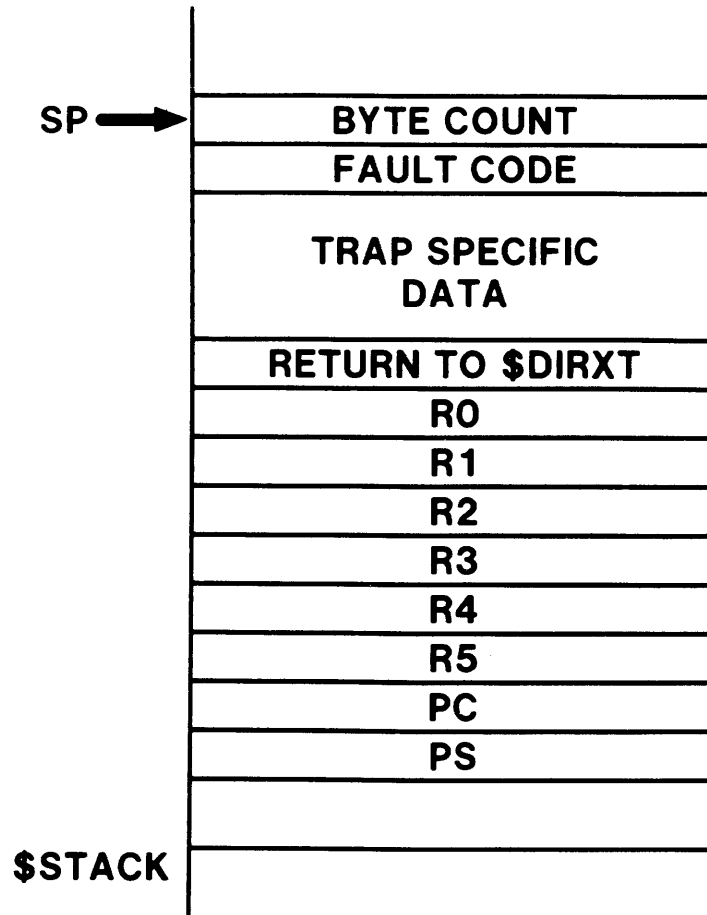
Figure 1Ø-3   The Stack After a System Crash - SST

Table 1Ø-2   Fault Codes for SSTs

Code	Number of Bytes	SST Trap
Ø	4	Traps through vector 4 (Odd address, etc.)
2	1Ø	Segment fault
4	4	Trace or break-point instruction
6	4	IOT instruction
1Ø	4	Illegal or reserved instruction
11	6	Non-RSX EMT trap instruction
16	14	Floating-point exception (11/4Ø)

```
        KERNEL Stack Pointer = 000562

Location            Contents

000556              024552  - PC at Crash

000560              000010  - PS at Crash        SST Information

000562*             000004  - Number of bytes

000564              000000  - SST Code

000566              003120  - Addr of $DIRXT

000570              000001  - R0

000572              070514  - R1

000574              074460  - R2

000576              053204  - R3                 Context at SST

000600              144000  - R4

000602              000000  - R5

000604              121204  - PC

000606              030000  - PS

000610              003126  - ADDR of $DIRXT

000612              000000  - R0

000614              070514  - R1

000616              074460  - R2                 Context at Trap

000620              053200  - R3

000622              144000  - R4

000624              000000  - R5

000626              121206  - PC

000630              170000  - PS
```

Example 10-1   A Sample KERNEL Stack

AN EXAMPLE `

Situation

● Privileged task TT2 is executing.

Task prompts for terminal number.

User enters terminal number.

System crashes.

CDA prints message:

Pool link error at 000040 -- FWD ptr = 000167
Size = 001736

Output from CDA

- Current task = RMDEMO

    Error indicates

        Error caused by another task
        Other task corrupted list of free pool blocks

- Contents of KERNEL stack indicate crash caused by IOT

    Consistent with pool pointer = 000167

    Note that 000167 is JMP instruction and 001736 is location of $CRASH

- Any recently or currently active privileged tasks may have caused crash

- Examination of listing of task TT2 shows that task allocates and uses a block of pool

    Address of pool stored at location PLACE in task

    Task map indicates address of PLACE is 121430

- Dump of task obtained from binary crash file

    Value of 054100 stored in PLACE

- Examination of pool dump indicates

    Value 000040 entered in 84 words starting at 054100

    Allocated block started at 054100 and contained 84 bytes

    Error at location 000630 in task: MOV should be MOVB

```
RSX-11M CRASH DUMP ANALYZER      V4.0       1-NOV-81   11:16        PAGE 1
VOLATILE REGISTERS


AFTER CRASH:   PS=000340    SP(K)=000660    SP(U)=121164

BEFORE CRASH:  PC=141362    PS=030000

R0=137034    R1=137034    R2=131120    R3=000000    R4=000167    R5=137014

MMR0=000001    MMR1=000000    MMR2=002326    MMR3=000060


              U S E R                          U N I B U S   M A P

     I   S P A C E        D   S P A C E        1   00000000
     PDR       PAR        PDR       PAR        2   00020000
     077506    000000     000000    000000     3   00040000
     077406    000200     000000    000000     4   00060000
     077406    000400     000000    000000     5   00100000
     077406    000600     000000    000000     6   00146200
     077406    001000     000000    000000     7   00166200
     077506    004732     000000    000000     8   17000000
     033006    005132     000000    000000     9   17000000
     077406    177600     000000    000000     10  17000000
                                               11  17000000
                                               12  17000000
          S U P E R V I S O R                  13  17000000
                                               14  17000000
     I   S P A C E        D   S P A C E        15  17000000
     PDR       PAR        PDR       PAR        16  17000000
     000000    000000     000000    000000     17  17000000
     000000    000000     000000    000000     18  17000000
     000000    000000     000000    000000     19  17000000
     000000    000000     000000    000000     20  17000000
     000000    000000     000000    000000     21  17000000
     000000    000000     000000    000000     22  17000000
     000000    000000     000000    000000     23  17000000
     000000    000000     000000    000000     24  17000000
                                               25  17000000
                                               26  17000000
           K E R N E L                         27  17000000
                                               28  17000000
     I   S P A C E        D   S P A C E        29  00045100
     PDR       PAR        PDR       PAR        30  17760000
     077506    000000     000000    000000     31  00105100
     077506    000200     000000    000000
     077506    000400     000000    000000
     077406    000600     000000    000000
     077506    001000     000000    000000
     077506    004732     000000    000000
     077406    005132     000000    000000
     077506    177600     000000    000000
```

Example 10-2   CDA - Volatile Registers

RSX-11M CRASH DUMP ANALYZER   V4.0   1-NOV-81   11:16   PAGE 2
KERNEL STACK

KERNEL STACK:

```
000400    000000  000000  000000  000000  000000  000000  000000  000000

000420    000000  000000  000000  000000  000000  000000  000000  000000

000440    000000  000000  000000  000000  000000  000000  000000  000000

000460    000000  000000  000000  000000  000000  000000  000000  000000

000500    000000  000000  000000  000000  000000  000000  000000  000000

000520    000000  000000  000000  000000  000000  000000  000000  000000

000540    000000  000000  000000  000000  000000  000000  000000  000000

000560    000000  000000  131514  131534  133054  133036  123156  001662

000600    006110  133304  044000  025312  132602  030000  026304  001000

000620    026304  001000  131542  160020  133040  010024  011604  124670

000640    000000  000002  000144  000000  036154  137014  141362  030000

000660    003236  000000  136256  131120  000000  136320  134554  142576

000700    170000
```

Example 10-3   CDA - KERNEL Stack

`.`

```
RSX-11M CRASH DUMP ANALYZER      V4.0       1-NOV-81    11:16        PAGE 3
SYSTEM COMMON


CRASH OCCURRED AT 11:12:27  1-NOV-81


CURRENT TASK = RMDEMO     TCB ADDRESS = 052110

$SYSID = 31      $EXSIZ = 120000     $SYSIZ = 9216./288K     $SYUIC = [1,54]

$STKDP = 000000      $COMEF:  <33-48> 000000  <49-64> 000000

SYSTEM NAME = RSX11M      $NTUIC = [1,54]

LOAD DEVICE = DR0      LBN = 00135325     FILE SIZE = 496.

SYSTEM FEATURE MASK (FIRST WORD) = 013377


        BIT SET              MEANING
        -------              -------
          EXT         22-BIT EXTENDED MEMORY SUPPORT
          MUP         MULTI-USER PROTECTION SUPPORT
          EXV         20K EXEC SUPPORTED
          DRV         LOADABLE DRIVER SUPPORT
          PLA         PLAS SUPPORT
          CAL         DYNAMIC CHECKPOINT SPACE ALLOCATION
          PKT         PREALLOCATION OF I/O PACKETS
          EXP         EXTEND TASK DIRECTIVE SUPPORTED
          OFF         PARENT/OFFSPRING TASKING SUPPORTED
          FDT         FULL DUPLEX TERMINAL DRIVER
          DYM         DYNAMIC MEMORY ALLOCATION SUPPORTED


SYSTEM FEATURE MASK (SECOND WORD) = 167400


        BIT SET              MEANING
        -------              -------
          DPR         DIRECTIVE PARTITION SUPPORT
          IRR         INSTALL, REQUEST, AND REMOVE TASK SUPPORT
          GGF         GROUP GLOBAL EVENT FLAG SUPPORT
          RAS         RECEIVE/SEND DATA PACKET SUPPORT
          RBN         ROUND ROBIN SCHEDULING SUPPORTED
          SWP         EXECUTIVE LEVEL DISK SWAPPING SUPPORTED
          STP         EVENT FLAG MASK IS IN THE TCB


SYSTEM FEATURE MASK (THIRD WORD) = 025214


        BIT SET              MEANING
        -------              -------
          CLI         MULTIPLE CLI SUPPORT
          EIS         SYSTEM REQUIRES THE EXTENDED INSTRUCTION SET
          STM         SYSTEM HAS SET SYSTEM TIME DIRECTIVE
          AST         SYSTEM HAS AST SUPPORT
```

Example 10-4   CDA - System Common
(Sheet 1 of 3)

RSX-11M CRASH DUMP ANALYZER   V4.0   1-NOV-81   11:16   PAGE 4
SYSTEM COMMON DUMP

ADDR	LABEL	VALUE	ADDR	LABEL	VALUE	ADDR	LABEL	VALUE
007750		010316	010110	$PRIHL	003100	010250		000000
		000207		$PRILL	001130			000000
	$HEADR	052654		$PFRSZ	000310			000000
		160400		$POLBP	000063			000000
	$COMEF	000000		$POLFL	000000			000000
		000000		$PARPT	017320			000000
	$SYSID	030463		$CLKHD	045716			000000
		020040		$COPT	043630			000000
007770	$TKNPT	110610	010130	$PARHD	117734	010270		000000
	$SHFPT	111040		$LDRPT	117510			000000
	$CKCNT	172542		$TSKHD	117510			000000
	$CKCSR	172540		$XCOM1	001200			000000
	$CKLDC	000144		$XCOM2	001345			000000
	$SYUIC	000454		$GGEF	000000			000000
		000000		$GFTCB	010150			000000
	$EXSIZ	120000		$GEFPT	010150		$ERBAF	000000
010010	$PWRFL	000000	010150	$GEFDM	000254	010310		000000
	$SIGFL	000000		$IDLCT	000001			000000
	$LOGHD	000000		$IDLPT	170017			000000
	$MCRCB	000000		$DYPMN	020035			000000
	$LSTLK	000000			020037			000000
		000003			020037			000000
	$CRAVL	046464			017440			000000
		000000			017440			000000
010030	$ACTHD	117510	010170		020040	010330		000000
	$DICSV	001037		$BTMSK	000001			000000
	$TKTCB	052110			000002			000000
	$LBUIC	000454			000004			000000
	$ABTIM	041233			000010			000000
	$RQSCH	000000			000020			000000
	$STKDP	000000			000040			000000
	$DEVHD	040746			000100			000000
010050	$RNDCT	000005	010210		000200	010350		000000
	$SWPCT	000036			000400		$ERFID	000000
	$ERRPT	000000			001000			000000
	$CFLPT	050740			002000			000000
		000000			004000			000000
		026354			010000			000000
	$INTCT	177777			020000		$PRMOD	000106
	$FRKHD	000000			040000		$SYSIZ	022000
010070		010066	010230		100000	010370		000000
	$FMASK	013377		$ERHEA	000000			135325
		167400			010232			051104
		025214		$ENTSQ	000000			000760
	$HFMSK	000003		$ERRSQ	000000			177777
	$FTTCB	112314		$ERFLA	000000			000015
	$PRISZ	033364			000000		$DPM	000037
	$POLST	000401			000000			000030

Example 10-4   CDA – System Common
(Sheet 2 of 3)

RSX-11M CRASH DUMP ANALYZER   V4.0   1-NOV-81   11:16   PAGE 5
SYSTEM COMMON DUMP

ADDR	LABEL	VALUE	ADDR	LABEL	VALUE	ADDR	LABEL	VALUE
010410		000074	010470	$PKAVL	054034	010550	$MCRPT	111750
		000074		$PKNUM	002403			050712
	$TKPS	000144		$MXEXT	177777			000000
		000121		$UMRHD	000000			000040
		000013		$UMRPT	170200			003404
		000001			000034			005015
		000013		$UMRWT	000000			000076
		000014			010504			005015
010430		000033	010510	$TEMP0	111750	010570		041515
	$TTNS	000063		$TEMP1	052514			037122
	$TLMTS	000200		$TEMP2	130562			000000
		000000		$TEMP4	000000			
		000014		$AVRHD	000000			
		000001		$MOULS	047064			
		000000		$CXDBL	000000			
		000001		$SYSNM	051522			
010450		000027	010530		030530			
		000000			046461			
		000073		$CLICQ	000000			
		000000			010534			
		000073		$CPTBL	010550			
		000000			052464			
	$TIKLM	000143			000000			
		000000			000000			

Example 10-4 . CDA - System Common
(Sheet 3 of 3)

```
RSX-11M CRASH DUMP ANALYZER     V4.0     1-NOV-81  11:16      PAGE 6
SYSTEM COMMON ALPHABETIZED DUMP


$ABTIM 041233      $ERLOF    000      $MOULS 047064      $STKDP 000000

$ACTHD 117510      $ERRPT 000000      $MXEXT 177777      $SWPCT 000036

$AVRHD 000000      $ERRSQ 000000      $NTUIC 000454      $SYSIZ 022000

$CFLPT 050740      $EXSIZ 120000      $PARHD 117734      $SYSNM 051522

$CKCNT 172542      $FMASK 013377      $PARPT 017320      $SYUIC 000454

$CKCSR 172540      $FRKHD 000000      $PFRSZ 000310      $TEMP0 111750

$CKLDC 000144      $GEFDM 000254      $PKAVL 054034      $TEMP1 052514

$CLICQ 000000      $GEFPT 010150      $PKMAX    005      $TEMP2 130562

$CLKHD 045716      $GFTCB 010150      $PKNUM    003      $TEMP4 000000

$COPT  043630      $GGEF  000000      $POLBP 000063      $TKNPT 110610

$CPTBL 010550      $HEADR 052654      $POLFL 000000      $TKPS  000144

$CRAVL 046464      $HFMSK 000003      $POLST 000401      $TKTCB 052110

$CURPR    341      $IDLCT    001      $PRIHL 003100      $TSKHD 117510

$CXDBL    000      $IDLFL    000      $PRILL 001130      $TTNS  000063

$DEVHD 040746      $IDLPT 170017      $PRISZ 033364      $UMRHD 000000

$DICSV 001037      $INTCT 177777      $PRMOD 000106      $UMRPT 170200

$DPM   000037      $LBUIC 000454      $PTTCB 112314      $UMRWT 000000

$ENTSQ 000000      $LDRPT 117510      $PWRFL 000000      $WTCSR 017320

$ERBAF    000      $LOGHD 000000      $RNDCT 000005      $WTDUM 017320

$ERFID 000000      $LSTLK 000000      $RQSCH 000000      $XCOM1 001200

$ERFLA    000      $MCRCB 000000      $SHFPT 111040      $XCOM2 001345

$ERHEA 000000      $MCRPT 111750      $SIGFL 000000
```

Example 10-5 CDA - System Common Alphabetized Dump

```
RSX-11M CRASH DUMP ANALYZER        V4.0        1-NOV-81   11:16        PAGE 7
POOL STATISTICS


    > > > > >   Pool link error at 000040 -- FWD ptr = 000167   Size = 001736

POOL SIZE (BYTES) = 21376.

LARGEST FREE BLOCK (BYTES) = 36.

TOTAL FREE BYTES = 144.

NUMBER OF FRAGMENTS = 13.

MINIMUM BLOCK SIZE (BYTES) = 4.


POOL BITMAP (CONSTRUCTED FROM LINKED POOL, BLOCK FREE IF BIT SET):

        000000  000000  060000  002014  000000  000000  000200  001000
        000000  001400  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000001  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  007600  002000  000000  000000  000000
        000000  170000  000037  000000  003400  000000  000000  000000
        017740  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
```

Example 1Ø-6   CDA - POOL Statistics
(Sheet 1 of 2)

RSX-11M CRASH DUMP ANALYZER   V4.0   1-NOV-81   11:16   PAGE 8
POOL STATISTICS

```
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000  000000  000000
        000000  000000  000000  000000  000000  000000
```

Example 10-6 CDA - POOL Statistics
(Sheet 2 of 2)

RSX-11M CRASH DUMP ANALYZER   V4.0   1-NOV-81   11:16   PAGE 19
ACTIVE TASKS

TT2

   TCB ADDRESS = 054440   PAR = GEN   PCB ADDRESS = 055174

   LOAD ADDRESS = 00440400   LOAD DEVICE = SY0:   LBN = 00117403

   PRI = 50.   I/O COUNT = 1.   UIC = [7,377]   TI = TT2:

   MAX SIZE = 000031   EVENT FLAGS = <1-16> 000000   <17-32> 000000

   T.STAT:

   T.ST2:   -CHK   WFR

   T.ST3:   -PMD   REM   PRV   MCR

   T.TIO: 000000

   TCB DUMP:

      000000   000000 000462 000000 100100 000000 000000 054452 000000
      000020   054456 000000 000000 044256 007754 000000 020001 074000
      000040   000062 117403 046166 055174 000031 111404 000000 000000
      000060   000031 047102 047102 000000 000000 000000 054532 000000
      000100   054536 000000 000001 054462 000000

            Example 10-7   CDA - Active Tasks: TT2

```
000040    000000  000000  055102  000001  126202  000000  000000  000000
000060    000376  000000  000000  000000  000000  055144  000007  046072
000100    000000  042360  000000  043630  000000  046166  000000  046072
000120    000000  046116  000000  046072  000000  000001  053654  120000
000140    132277  046430  000123  000000  000612  051006  170000  124636
000160    000000  177777  044000  000000  000056  120533  000000
```

TT2

HEADER ADDRESS = 054554   TCB ADDRESS = 054440

PS=170000   PC=122200

R0=000000   R1=054350   R2=044336   R3=000001   R4=000001   R5=000060   SP=121172

INITIAL PS = 170017   INITIAL PC = 121442   INITIAL SP = 121172

HEADER SIZE = 122.   NO. OF WINDOWS = 1.   NO. OF LUNS = 6.

CURRENT UIC = [7,377]   DEFAULT UIC = [7,377]

H.WND = 054702   H.GARD = 054744   H.VEXT = 000000   H.SPRI = -5.

DSW = 000001   H.FCS = 000000   H.FORT = 000000   H.OVLY = 000000

LOGICAL UNIT TABLE:

#	DEV	WINDOW	W.CTL	W.FCB	F.FNUM	F.FSEQ	F.STAT	NAC	NLCK
1	SYO:	000000							
2	SYO:	000000							
3	SYO:	000000							
4	SYO:	000000							
5	TIO:	000000							
6	CLO:	000000							

WINDOW BLOCKS:

PAR	VIRT LIMITS	ATT DESC	WND SIZE	OFFSET	1ST PDR	NO.	LAST PDR
GEN	120000 123077	047074	000031	000000	177612	1	014006

HEADER:

```
000000     121172  000172  053254  135600  003777  003777  170017  121442
```

Example 10-8 CDA - Tasks Headers: TT2
(Sheet 1 of 2)

RSX-11M CRASH DUMP ANALYZER   V4.0   1-NOV-81   11:16   PAGE 31
TASK HEADERS

```
000020   121172  000000  000000  000000  000000  000000  000000  000000
000040   000000  000000  054702  000001  000000  000000  000000  000000
000060   000373  000000  000000  000000  000000  054744  000006  046166
000100   000000  046166  000000  046166  000000  046166  000000  046072
000120   000000  046116  000000  000001  055174  120000  123077  047074
000140   000031  000000  000612  014006  170000  122200  000060  000001
000160   000001  044336  054350  000000  000000
```

Example 10-8   CDA - Tasks Headers: TT2
(Sheet 2 of 2)

```
RSX-11M CRASH DUMP ANALYZER      V4.0      1-NOV-81   11:16        PAGE 82
POOL DUMP

053260   * 114250 * 053276 * 000001 * 115404   !XQ    M5F    A X2D!!( >V      !
053270   * 000000 * 000106 * 114250 * 000000   !      A0 XQ      !!   F (      !
053300   * 000401 * 115404 * 000000 * 001000   ! FQ X2D         L2!!          !
053310   * 000016 * 121350 * 000000 * 000000   !  N ZBX             !!  h"     !
053320   * 000401 * 000000 * 000000 * 000000   ! FQ                 !!         !
053330   * 114730 * 121246 * 000000 * 000000   !XXX Z 8            !!X &"       !
053340   * 066117 * 131574 * 000000 * 000000   !QMG ...            !!01!3      !
053350   * 000033 * 000000 * 000000 * 000000   !  $                !!          !
053360     053640   000044   111404   041740   !N 2    6 WT6 J4P!! W$     `C!
053370     000000   000000   000001   111426   !          A WUN!!              !
053400     053640   000024   000000   000000   !N 2    T           !! W        !
053410     000403   000000   053640   000010   ! FS       N 2   H!!      W    !
053420     000000   000000 * 120412 * 000212   !          YOZ  CR!!      !  !
053430   * 143107 * 000000 * 000454 * 000454   !1.9        GT   GT!!GF  , , !
053440   * 170017 * 120750 * 120412 * 000000   !8PO Y6H YOZ      !! ph! !  !
053450   * 000000 * 000000 * 000000 * 000000   !                   !!         !
053460   * 000000 * 000000 * 000000 * 000000   !                   !!         !
053470   * 053552 * 000002 * 143030 * 000000   !M9R    B 1$2       !!JW    F  !
053500   * 143142 * 000000 * 000373 * 000000   !1/Z        FK      !!bF  {   !
053510   * 000000 * 000000 * 000000 * 053634   !              N .!!       W!
053520   * 000006 * 046166 * 000000 * 043630   !  F LIN        KRH!!  vL    G!
053530   * 000000 * 041006 * 000000 * 046166   !      JVV      LIN!!   B  vL!
053540   * 000000 * 046072 * 000000 * 046116   !      LG4      LHN!!  :L  NL!
053550   * 000000 * 000002 * 046674 * 120000   !      B LQT YX !!     <M   !
053560   * 144277 * 052360 * 000243 * 000000   !2D1 MWX   DC     !!?HpT#     !
053570   * 001212 * 021006 * 115404 * 160000   ! PJ EQO X2D 53X!!   "     `!
053600   * 175577 * 053270 * 000156 * 000147   ! IW M5    BO   BW!! {8Vn s !
053610   * 000616 * 066402 * 170000 * 121016   ! I8 QQZ 8P   Y7F!!    m p "!
053620   * 000000 * 000000 * 143107 * 000060   !            1.9  AH!!     GFO !
053630   * 000060 * 120431 * 000000 * 000000   ! AH Y1A         !!0  !     !
053640     054224   000014   000000   000000   !NF6    L          !! X       !
053650     000403   000000 * 054374 * 000062   ! FS       NIT  AJ!!     !X2 !
053660   * 026226 * 000000 * 055174 * 115274   !GEN       NSD XOL!! ,  !Z< !
053670   * 004076 * 000123 * 000000 * 000123   !ALO  BC      BC!!> S   S !
053700   * 000000 * 111404 * 000050 * 054750   !      WT6  A  NOP!!    ( hY!
053710   * 000000 * 046430 * 046430 * 000000   !      LMP LMP    !!  M M  !
053720   * 000000 * 000062 * 000000 * 131574   !      AJ      ...!!  2   !3!
053730   * 007537 * 000000 * 053732 * 000000   !BRO       NBJ   !!_   ZW  !
053740   * 053736 * 177776 * 000001 * 044000   !NBN  8N    A KT2!!~W~    H!
053750   * 054440 * 100400 * 020000 * 050000   !NJP TYX ED2 L2 !! Y     P!
053760   * 000062 * 142440 * 046142 * 115274   ! AJ 1UX LH4 XOL!!2  EbL< !
053770   * 000306 * 054440 * 000000 * 000000   ! D8 NJP         !!F   Y    !
054000   * 000306 * 000000 * 054002 * 000000   ! D8       NCJ   !!F    X  !
054010   * 000000 * 000000 * 054012 * 000000   !          NCR   !!    X  !
054020   * 054016 * 000000 * 000003 * 053742   !NCV       C NBR!! X   bW!
054030   * 000000 * 000000 + 055240 + 000044   !          NT   6!!    Z$ !
054040   + 000020 + 000000 + 044416 + 000400   ! P       K$V  FP!!    I  !
054050   + 135154 + 005103 + 140054 + 000000   !/3D AY$ 0/6     !!!1:C ,@  !
054060   + 005077 + 140144 + 000070 + 000000   !AYW 01L  AP     !!? d@8     !
054070   + 000000 + 000000 + 000000 + 000000   !                   !!        !
054100   * 000040 * 000040 * 000040 * 000040   !  2   2   2   2!!            !

              [ABOVE LINE REPEATED 9. TIMES]

054220   * 000040 * 000040   000040   000040   !  2   2   2  2 2!!            !
054230     000040   000040   000040   000040   !  2   2   2  2 2!!            !
```

Example 10-9 CDA - POOL Dump
(Sheet 1 of 2)

```
054240        000040        000040        000040        000040      !  2    2    2    2!!              !
054250        000040        000040        000040        000040      !  2    2    2    2!!              !
054260        000040        000040  * 000040  * 000040      !  2    2    2    2!!              !
054270    * 000040  * 000040  * 000040  * 000040      !  2    2    2    2!!              !
054300    * 000040  * 000040  * 000040  * 000040      !  2    2    2    2!!              !
054310    * 000040  * 000040  * 000040  * 000040      !  2    2    2    2!!              !
054320    * 000040  * 000040  * 000040  * 000040      !  2    2    2    2!!              !
054330    * 000040  * 000040  * 000040  * 000040      !  2    2    2    2!!              !
054340    * 000040  * 000040  * 000040  * 000040      !  2    2    2    2!!     P  *b/   !
054350    * 170000  * 125200  * 127542  * 000000      !8P    $KH  .BR      !!  P  *b/   !
054360    * 000004  * 130034  * 131100  * 000110      !  D   .GD  .T2   A2!!     0@2H   !
054370    * 000000  * 000000  * 000000  * 000773      !               L$!!         {   !
054400    * 111404  * 055050  * 044000  * 000540      !WT6  NQ    KT2   H2!!     (Z H`  !
054410    * 121246  * 004110  * 140046  *.000000      !Z 8  AM    0/0      !!&"H &@     !
054420    * 000005  * 140111  * 000056  * 000000      !  E  OOY   AF       !!    I@.    !
054430    * 000000  * 100000  * 000000  * 053654      !     TSH         NAD!!       ,W!
054440    * 000000  * 000462  * 000000  * 100100      !     GZ          TT2!!  2    @  !
054450    * 000000  * 000000  * 054452  * 000000      !           NJZ      !!    *Y    !
054460    * 054456  * 000000  * 000000  * 044256      !NJO            KYF!!.Y       .H!
054470    * 007754  * 000000  * 020001  * 074000      !BU6        ED3  SH !!1        x!
054500    * 000062  * 117403  * 046166  * 055174      ! AJ  YQ$   LIN  NSD!!2    vL!Z!
054510    * 000031  * 111404  * 000000  * 000000      !  Y  WT6           !!          !
054520    * 000031  * 047102  * 047102  * 000000      !  Y  LT4   LT4     !!  BNBN    !
054530    * 000000  * 000000  * 054532  * 000000      !           NK4     !!    ZY    !
054540    * 054536  * 000000  * 000001  * 054462      !NK8          A  NJ4!!^Y    2Y!
054550    * 000000  * 001000  * 121172  * 000172      !     L2  Y94   CB!!    z"z    !
054560    * 053254  * 135600  * 003777  * 003777      !M4.  0    AKG  AKG!!,V ;     !
054570    * 170017  * 121442  * 121172  * 000000      !8PO  ZDB  Y94     !! P"#z"   !
054600    * 000000  * 000000  * 000000  * 000000      !                  !!         !
054610    * 000000  * 000000  * 000000  * 000000      !                  !!         !
054620    * 054702  * 000001  * 000000  * 000000      !NNR    A          !!BY       !
054630    * 000000  * 000000  * 000373  * 000000      !               FK !!     {   !
054640    * 000000  * 000000  * 000000  * 054744      !               NOL!!      dY!
054650    * 000006  * 046166  * 000000  * 046166      ! F LIN        LIN!!  vL  vL!
054660    * 000000  * 046166  * 000000  * 046166      !   LIN        LIN!!  vL  vL!
054670    * 000000  * 046072  * 000000  * 046116      !   LG4        LHN!!  :L  NL!
054700    * 000000  * 000001  * 055174  * 120000      !      A  NSD  YX !!     !Z   !
054710    * 123077  * 047074  * 000031  * 000000      !ZW9  LT.    Y     !!?&<N     !
054720    * 000612  * 014006  * 170000  * 122200      ! I4  C30  8P   ZL2!!    P  $!
054730    * 000060  * 000001  * 000001  * 044336      ! AH    A    A  KZN!!O     ^H!
054740    * 054350  * 000000  * 000000  * 120424      !NI            Y06!!hX     !!
054750    * 120502  * 000176  * 140354  * 135600      !Y2B  CF   04.  0  !!B!~ 1@ ;!
054760    * 000454  * 000454  * 170017  * 121714      ! GT   GT  8PO  ZHL!!, ,  PL#!
054770    * 120506  * 000000  * 000000  * 000000      !Y2F              !!F!       !
055000    * 000000  * 000000  * 000000  * 000000      !                  !!         !
055010    * 000000  * 000000  * 055102  * 000001      !           NQZ   A!!    BZ   !
055020    * 126202  * 000000  * 000000  * 000000      !$XB              !! ,       !
055030    * 000376  * 000000  * 000000  * 000000      ! FN              !!~        !
055040    * 000000  * 055144  * 000007  * 046072      !     NRT    G  LG4!!   dZ  :L!
055050    * 000000  * 042360  * 000000  * 043630      !     KAH      KRH!!  PD   G!
055060    * 000000  * 046166  * 000000  * 046072      !     LIN      LG4!!  vL  :L!
055070    * 000000  * 046116  * 000000  * 046072      !     LHN      LG4!!  NL  :L!
055100    * 000000  * 000001  * 053654  * 120000      !      A  NAD  YX !!    ,W  !
055110    * 132277  * 046430  * 000123  * 000000      !.61  LMP    BC    !!?4 MS   !
055120    * 000612  * 051006  * 170000  * 124636      ! I4  MD8  8P   $EV!!    R P )!
055130    * 000000  * 177777  * 044000  * 000000      !      80  KT2     !!     H   !
```

Example 10-9   CDA - POOL Dump
(Sheet 2 of 2)

```
 1                                              .TITLE   DUMP
 2                                       ;+
 3                                       ;
 4                                       ;    THIS TASK ASKS THE USER FOR A TERMINAL NUMBER AND A
 5                                       ;    COMMAND LINE.   IT THEN PASSES THE COMMAND TO MCR
 6                                       ;    AS IF HAD ORIGINATED AT THE SPECIFIED TERMINAL.   IT
 7                                       ;    PERFORMS THIS TASK BY USING SYSTEM SUBROUTINES, $ALOCB
 8                                       ;    AND $QMCRL, TO ALLOCATE A BLOCK FROM POOL AND PASS IT
 9                                       ;    TO MCR.
10                                       ;
11                                       ;    THE TASK MUST BE BUILT PRIVILEGED AND MAP TO THE EXECUTIVE.
12                                       ;
13                                       ;    TO ASSEMBLE:
14                                       ;
15                                       ;    >MAC TERM=[1,1]EXEMC/ML,[11,10]RSXMC/PA:1,[(YOUR UIC)]TERM
16                                       ;
17                                       ;    TO BUILD:
18                                       ;
19                                       ;    >TKB TERM/PR:5=TERM,[1,54]RSX11M.STB/SS
20                                       ;
21                                       ;+
22                                              .MCALL   EXIT$S,QIOW$S,DCBDF$
23 000000                                       DCBDF$
24                                       ;
25                                       ;    TERMINAL MESSAGES
26                                       ;
27 000000      012    015    105  PRMT:  .ASCII   <12><15>/ENTER A TERMINAL NUMBER: /      ;PROMPT
   000003      116    124    105
   000006      122    040    101
   000011      040    124    105
   000014      122    115    111
   000017      116    101    114
   000022      040    116    125
   000025      115    102    105
   000030      122    072    040
28             000033                    PLEN   =        .-PRMT                         ;LENGTH OF PROMPT
29 000033      105    116    124  MES1:  .ASCII   /ENTER A COMMAND FOR MCR/
   000036      105    122    040
   000041      101    040    103
   000044      117    115    115
   000047      101    116    104
   000052      040    106    117
   000055      122    040    115
   000060      103    122
30             000027                    LENT1  =        .-MES1
31 000062      116    125    115  MESS1: .ASCII   /NUMBER IS NOT OCTAL, TRY AGAIN!/
   000065      102    105    122
   000070      040    111    123
   000073      040    116    117
   000076      124    040    117
   000101      103    124    101
   000104      114    054    040
   000107      124    122    131
   000112      040    101    107
   000115      101    111    116
   000120      041
32             000037                    LEN1   =        .-MESS1
```

Example 10-10   Listing of Task TT2 (DUMP)
(Sheet 1 of 5)

```
33 000121     124      105     122   ERMES:  .ASCII  /TERMINAL NOT FOUND!/
   000124     115      111     116
   000127     101      114     040
   000132     116      117     124
   000135     040      106     117
   000140     125      116     104
   000143     041
34            000023                  ERL     =       .-ERMES
35 000144     116      117     040   POOL:   .ASCII  /NO POOL AVAILABLE!/
   000147     120      117     117
   000152     114      040     101
   000155     126      101     111
   000160     114      101     102
   000163     114      105     041
36            000022                  PL      =       .-POOL
37 000166     111      057     117   IOERR:  .ASCII  !I/O ERROR ON COMMAND INPUT!
   000171     040      105     122
   000174     122      117     122
   000177     040      117     116
   000202     040      103     117
   000205     115      115     101
   000210     116      104     040
   000213     111      116     120
   000216     125      124
38            000032                  IOL     =       .-IOERR
39                                            .EVEN                           ;MAKE SURE OF WORD BOUNDARY
40                                    ;
41                                    ;       PROGRAM DATA
42                                    ;
43 000220                             BUF:    .BLKB   4                       ;BUFFER FOR TERMINAL NUMBER
44 000224                             IOSB:   .BLKW   2                       ;I/O STATUS BLOCK
45 000230     124      124            NAM:    .ASCII  /TT/                    ;TERMINAL NAME
46 000232     177777                  ERRCNT: .WORD   -1                      ;ERROR INDICATOR
47 000234     000000                  COMM:   .WORD   0                       ;ADDRESS OF COMMAND LINE IN POOL
48 000236     000000                  PLACE:  .WORD   0                       ;ADDRESS OF POOL BLOCK
49 000240     000000                  UCB:    .WORD   0                       ;SAVED UCB ADDRESS
50 000242     000000                  DCB:    .WORD   0                       ;SAVED DCB ADDRESS
51 000244     000000                  LOW:    .WORD   0                       ;SAVED LOW UNIT NUMBER
52 000246     000000                  HIGH:   .WORD   0                       ;SAVED HIGH UNIT NUMBER
53                                    ;
54                                    ;       BEGINNING OF CODE
55                                    ;
56 000250                             TERM:   QIOW$S  #IO.WVB,#5,#1,,,,<#PRMT,#PLEN,#40>      ;PROMPT
57 000322                                     QIOW$S  #IO.RVB,#5,#1,,,#IOSB,,<#BUF,#4> ;GET 'TT' NUMBER
58 000374     016703   177626                 MOV     IOSB+2,R3               ;SAVE CHARACTER COUNT
59 000400     012700   000220'                MOV     #BUF,R0                 ;SET UP FOR CALL TO CONVERSION
60 000404     005002                           CLR     R2                     ; ROUTINE - FROM ASCII TO
61 000406                                      CALL    $COTB                  ;   BINARY(OCTAL)
62                                                                            ; R1 WILL CONTAIN CONVERTED NUMBER
63                                                                            ; R2 WILL CONTAIN TERMINATOR
64 000412     005702                           TST     R2                     ;TEST FOR ERROR
65 000414     001402                           BEQ     CONT                   ;IF NOT CONTINUE
66 000416     000167   000436                  JMP     ERR1                   ;GO TO ERROR ROUTINE
67 000422                             CONT:    CALL    $SWSTK,BACK            ;;SWITCH TO SYSTEM STATE
68 000426     012700   000000G                MOV     ##DEVHD,R0             ;;PICK UP LISTHEAD FOR DCB LIST
69 000432     011000                  L1:     MOV     (R0),R0                ;;GO TO NEXT DCB
70 000434     001422                           BEQ     GO                     ;;IF NONE FINISH UP
```

Example 10-10 Listing of Task TT2 (DUMP)
(Sheet 2 of 5)

278

SYSTEM CRASHES

```
71 000436  026760  177566  000004          CMP    NAM,D.NAM(RO)         ;;LOOK FOR 'TT' DCB
72 000444  001372                           BNE    L1                    ;;IF NOT 'TT' GET NEXT DCB
73 000446  116067  000006  177570          MOVB   D.UNIT(RO),LOW        ;;GET LOW UNIT NUMBER
74 000454  116067  000007  177564          MOVB   D.UNIT+1(RO),HIGH     ;;GET HIGH UNIT NUMBER
75 000462  020167  177560                  CMP    R1,HIGH               ;;COMPARE INPUT WITH HIGH
76 000466  003361                           BGT    L1                    ;;IF INPUT LARGER GET NEXT DCB
77 000470  010067  177546                  MOV    RO,DCB                ;;SAVE DCB ADDRESS
78 000474  062767  000001  177530          ADD    #1,ERRCNT             ;;INDICATE NO ERROR HERE
79 000502                          GO:     RETURN                       ;;RETURN TO USER STATE
80 000504                          BACK:                                ;NOW IN USER STATE
81 000504  005767  177522                  TST    ERRCNT                ;TEST FOR ERROR
82 000510  001402                           BEQ    1$                    ;CONTINUE IF OK
83 000512  000167  000430                  JMP    ERR2                  ;GO TO ERROR ROUTINE
84 000516  016700  177520          1$:     MOV    DCB,RO                ;GET DCB ADDRESS
85 000522  016002  000002                  MOV    D.UCB(RO),R2          ;GET ADDRESS OF FIRST UCB
86 000526  010104                           MOV    R1,R4                 ;MOVE FOR CALCULATIONS
87 000530  166704  177510                  SUB    LOW,R4                ;FIND THE NUMBER WITHIN UCBS
88                                                                        ;  FOR CURRENT DCB
89 000534  016005  000010                  MOV    D.UCBL(RO),R5         ;GET UCB LENGTH
90 000540  070504                           MUL    R4,R5                 ;CALCULATE OFFSET FOR UCB
91 000542  060502                           ADD    R5,R2                 ;CALCULATE BEGINNING OF UCB
92 000544  010267  177470                  MOV    R2,UCB                ;SAVE UCB ADDRESS
93 000550  012767  177777  177454          MOV    #-1,ERRCNT            ;RESET ERROR COUNT
94 000556                                   CALL   $SWSTK,BACK2          ;;SWITCH TO SYSTEM STATE
95 000562  012701  000124                  MOV    #84.,R1               ;;SET BLOCK SIZE
96 000566                                   CALL   $ALOCB                ;;ALLOCATE BLOCK FROM POOL
97 000572  103404                           BCS    NONE                  ;;NO ROOM IN POOL
98 000574  005267  177432                  INC    ERRCNT                ;;INDICATE NO ERROR HERE
99 000600  010067  177432                  MOV    RO,PLACE              ;;SAVE ADDRESS OF POOL BLOCK
100 000604                         NONE:    RETURN                       ;;RETURN TO USER STATE
101 000606                         BACK2:                                ;BACK IN USER STATE
102 000606  005767  177420                  TST    ERRCNT                ;CHECK FOR ERROR
103 000612  001402                           BEQ    1$                    ;CONTINUE IF OK
104 000614  000167  000406                  JMP    ERR3                  ;GO TO ERROR ROUTINE
105 000620  012700  000124          1$:     MOV    #84.,RO               ;SET LENGTH OF BLOCK
106 000624  016701  177406                  MOV    PLACE,R1              ;GET ADDRESS OF BLOCK
107 000630  012721  000040          2$:     MOV    #40,(R1)+             ;PLACE BLANKS IN BLOCK
108 000634  077003                           SOB    RO,2$                 ;LOOP UNTIL THROUGH
109 000636  016767  177374  177370          MOV    PLACE,COMM            ;CALCULATE LOCATION OF COMMAND
110 000644  062767  000004  177362          ADD    #4,COMM               ;  LOCATION IN BLOCK
111 000652                                   QIOW$S  #IO.WVB,#5,#1,,,,<#MES1,#LENT1,#40>    ;PROMPT FOR COMMAND
112 000724  005067  177274                  CLR    IOSB                  ;CLEAR THE I/O STATUS - WORD 1
113 000730  005067  177272                  CLR    IOSB+2                ;   "     "      "     "   - WORD 2
114 000734                                   QIOW$S  #IO.RVB,#5,#1,,IOSB,,<COMM,#80.>        ;INPUT COMMAND
115 001006  103002                           BCC    3$                    ;CONTINUE IF OK
116 001010  000167  000272                  JMP    ERR4                  ;BRANCH IF QIO ERROR
117 001014  105767  177204          3$:     TSTB   IOSB                  ;CHECK FOR I/O ERROR
118 001020  002002                           BGE    4$                    ;CONTINUE IF OK
119 001022  000167  000260                  JMP    ERR4                  ;BRANCH IF ERROR
120 001026                         4$:      CALL   $SWSTK,BACK3          ;;SWITCH TO SYSTEM STATE
121 001032  016701  177200                  MOV    PLACE,R1              ;;GET ADDRESS OF BLOCK
122 001036  016761  177176  000002          MOV    UCB,2(R1)             ;;PLACE UCB ADDRESS IN BLOCK
123 001044                                   CALL   $QMCRL                ;;QUEUE BLOCK TO MCR
124 001050                                   RETURN                       ;;LEAVE SYSTEM STATE
125 001052                         BACK3:                                ;BACK IN USER STATE
126 001052                                   EXIT$S                       ;EXIT
127                                          ;
```

<div align="center">

Example 10-10 Listing of Task TT2 (DUMP)
(Sheet 3 of 5)

</div>

```
DUMP    MACRO M1200  01-NOV-81 11:09  PAGE 3-3

128                              ;      ROUTINE TO HANDLE BAD TERMINAL NUMBER
129                              ;
130 001060             ERR1:     QIOW$S  #IO.WVB,#5,#1,,,,<#MESS1,#LEN1,#40>      ;PRINT MESSAGE
131 001132  012700  000220'      MOV     #BUF,RO                    ;GET INPUT BUFFER ADDRESS
132 001136  005020    CLEAR:     CLR     (RO)+                      ;CLEAR BUFFER
133 001140  077302               SOB     R3,CLEAR                   ;R3 CONTAINS CHARACTER COUNT
134 001142  000167  177102       JMP     TERM                       ;TRY AGAIN
135                              ;
136                              ;      ROUTINE TO HANDLE NON-EXISTENT TERMINAL
137                              ;
138 001146             ERR2:     QIOW$S  #IO.WVB,#5,#1,,,,<#ERMES,#ERL,#40>       ;PRINT MESSAGE
139 001220               EXIT$S                                     ;EXIT
140                              ;
141                              ;      ROUTINE TO HANDLE POOL ALLOCATION FAILURE
142                              ;
143 001226             ERR3:     QIOW$S  #IO.WVB,#5,#1,,,,<#POOL,#PL,#40>         ;PRINT MESSAGE
144 001300               EXIT$S                                     ;EXIT
145                              ;
146                              ;      ROUTINE TO HANDLE I/O ERRORS ON COMMAND INPUT
147                              ;      THE POOL BLOCK MUST BE DEALLOCATED!
148                              ;
149 001306             ERR4:     QIOW$S  #IO.WVB,#5,#1,,,,<#IOERR,#IOL,#40>       ;PRINT MESSAGE
150 001360               CALL    $SWSTK,FIN                 ;;SWITCH TO SYSTEM STATE
151 001364  016700  176646       MOV     PLACE,RO                   ;;GET ADDRESS OF BLOCK
152 001370  012701  000124       MOV     #84.,R1                    ;;SET BLOCK LENGTH
153 001374               CALL    $DEACB                     ;;DEALLOCATE BLOCK
154 001400               RETURN                             ;;LEAVE SYSTEM STATE
155 001402             FIN:                                 ;BACK IN USER STATE
156 001402               EXIT$S                             ;EXIT
157         000250'      .END    TERM
```

Example 10-10 Listing of Task TT2 (DUMP)
(Sheet 4 of 5)

```
DUMP    MACRO M1200  01-NOV-81 11:09  PAGE 3-4
SYMBOL TABLE
```

A$$BIO= 000000	D.MSK 000014	K$$IEN= 000113	PRMT 000000R	T$$BTW= 000000
A$$BRT= 000000	D.NAM 000004	K$$LDC= 000144	P$$BPR= 000063	T$$BUF= 000000
A$$CHK= 000000	D.PCB 000034	K$$TPS= 000144	P$$CTL= 000000	T$$CCA= 000000
A$$CLI= 000004	D.UCB 000002	LD$CO = 000000	P$$FRS= 000310	T$$CCO= 000000
A$$CPS= 000000	D.UCBL 000010	LD$DD = 000000	P$$GMX= 000000	T$$CPW= 000000
A$$NSI= 000000	D.UNIT 000006	LD$DK = 000000	P$$HIL= 003100	T$$CTR= 000000
A$$PRI= 000000	D.VCAN= 000002	LD$DL = 000000	P$$LAS= 000000	T$$CUP= 000000
A$$TRP= 000000	D.VDEB= 177776	LD$DM = 000000	P$$LOL= 001130	T$$C11= 000001
BACK 000504R	D.VINI= 000000	LD$DR = 000000	P$$OFF= 000000	T$$ESC= 000000
BACK2 000606R	D.VOUT= 000004	LD$DS = 000000	P$$P45= 000000	T$$GMC= 000000
BACK3 001052R	D.VFWF= 000006	LD$DT = 000000	P$$RFL= 000000	T$$GTS= 000000
BUF 000220R	ERL = 000023	LD$LP = 000000	P$$RTY= 000000	T$$HFF= 000000
B$$LV1= 030463	ERMES 000121R	LD$MM = 000000	P$$SRF= 000000	T$$HLD= 000000
B$$LV2= 020040	ERRCNT 000232R	LD$TT = 000000	Q$$OPT= 000005	T$$J16= 000001
CLEAR 001136R	ERR1 001060R	LENT1 = 000027	R$$DER= 000000	T$$KMG= 000000
COMM 000234R	ERR2 001146R	LEN1 = 000037	R$$EXV= 000000	T$$LWC= 000000
CONT 000422R	ERR3 001226R	LOW 000244R	R$$JS1= 000001	T$$RED= 000000
C$$CDA= 000002	ERR4 001306R	L$$ASG= 000000	R$$K11= 000001	T$$RNE= 000000
C$$CHE= 000000	E$$ACT= 000000	L$$DRV= 000000	R$$LKL= 000001	T$$RPR= 000000
C$$CKP= 000004	E$$DVC= 000000	L$$LDR= 000000	R$$L11= 000001	T$$RST= 000000
C$$CSR= 177404	E$$LOG= 000000	L$$PTO= 000036	R$$MOF= 000000	T$$RUB= 000000
C$$INT= 000000	E$$XPR= 000000	L$$P11= 000001	R$$M11= 000001	T$$SMC= 000000
C$$ONS= 000001	FIN 001402R	L1 000432R	R$$NDC= 000005	T$$SYN= 000000
C$$ORE= 002022	F$$LPP= 000000	MESS1 000062R	R$$NDH= 000226	T$$TRW= 000000
C$$OTM= 000000	F$$LVL= 000001	MES1 000033R	R$$NDL= 000001	T$$UTB= 000000
C$$RSH= 177564	GO 000502R	M$$CRB= 000124	R$$POI= 000000	T$$UTO= 000170
C$$RUN= 000000	G$$EFN= 000000	M$$CRX= 000000	R$$SND= 000000	T$$U58= 000001
C$$SMT= 000000	G$$TPP= 000000	M$$EIS= 000000	R$$11M= 000000	T$$VBF= 000000
C$$TTY= 177564	G$$TSS= 000000	M$$EXT= 000000	R$$60F= 000000	T$$30P= 000000
DCB 000242R	G$$TTK= 000000	M$$FCS= 000000	R$$611= 000001	UCB 000240R
D$$H11= 000001	HIGH 000246R	M$$MGE= 000000	S$$ECC= 000000	U$$MHI= 000000
D$$IAG= 000000	H$$RTZ= 000074	M$$MUP= 000000	S$$NM1= 051522	U$$MLO= 160000
D$$ISK= 000000	IOERR 000166R	M$$OVR= 000000	S$$NM2= 030530	U$$MRN= 170234
D$$L11= 000002	IOL = 000032	NAM 000230R	S$$NM3= 046461	V$$CTR= 000400
D$$M11= 000001	IOSB 000224R	NONE 000604R	S$$TIM= 000000	V$$RSN= 000040
D$$PAR= 000000	IO.RVB= ****** GX	N$$LDV= 000001	S$$TOP= 000000	X$$DBT= 000000
D$$SHF= 000000	IO.WVB= ****** GX	N$$MOV= 000041	S$$WLK= 000000	$ALOCB= ****** GX
D$$WCK= 000000	I$$RAR= 000000	N$$UMR= 000034	S$$WPC= 000036	$COTB = ****** GX
D$$YNC= 000000	I$$RDN= 000000	PL = 000022	S$$WPR= 000005	$DEACB= ****** GX
D$$YNM= 000000	K$$AST= 000000	PLACE 000236R	S$$YSZ= 022000	$DEVHD= ****** GX
D.DSP 000012	K$$CNT= 172542	PLEN = 000033	TERM 000250R	$QMCRL= ****** GX
D.LNK 000000	K$$CSR= 172540	POOL 000144R	T$$ACR= 000000	$$$ARG= 000002

```
. ABS.  000036    000
        001410    001
ERRORS DETECTED:  0

VIRTUAL MEMORY USED:  10113 WORDS  ( 40 PAGES)
DYNAMIC MEMORY:  16120 WORDS  ( 62 PAGES)
ELAPSED TIME:  00:00:13
DUMP,DUMP/-SP=[1,1]EXEMC/ML,[11,10]RSXMC/PA:1,[7,377]DUMP
```

Example 10-10 Listing of Task TT2 (DUMP)
(Sheet 5 of 5)

Partition name : GEN
Identification : M4.0
Task   UIC   : [7,377]
Stack   limits: 120172 121171 001000 00512.
PRG xfr address: 121442
Task attributes: PR
Total address windows: 1.
Task   image   size   : 800. words
Task address limits: 120000 123007
R-W disk blk limits: 000002 000005 000004 00004.

*** Root segment: DUMP

R/W mem   limits: 120000 123007 003010 01544.
Disk blk limits: 000002 000005 000004 00004.

Memory allocation synopsis:

Section		Title	Ident	File
. BLK.:(RW,I,LCL,REL,CON)	121172 001504 00836.			
	121172 001410 00776.	DUMP	M4.0	DUMP.OBJ;2
$$RESL:(RO,I,LCL,REL,CON)	122676 000112 00074.			

Global symbols:

IO.RVB 010400   IO.WVB 011000 $ALOCB 011346 $DEACB 011614 $DEVHD 010046 $QMCRL 021432

*** Task builder statistics:

    Total work file references: 850.
    Work   file   reads: 0.
    Work   file writes: 0.
    Size of core pool: 20506. words (80. pages)
    Size of work file: 768. words (3. pages)

    Elapsed time:00:00:04

        Example 10-11   Map of Task TT2

SYSTEM CRASHES

TASK DUMP OF TT2

```
TCB ADDRESS = 054440        HEADER ADDRESS = 054554
```

WINDOW BLOCKS:

PAR	VIRT LIMITS	ATT DESC	WND SIZE	OFFSET	1ST PDR	NO.	LAST PDR
GEN	120000 123077	047074	000031	000000	177612	1	014006

WINDOW #1 -- TASK VIRTUAL LIMITS 120000-123077

PHYSICAL STARTING ADDRESS = 00440400

```
120000    000000  000172  053254  135600     !      CB M4. O  !!   z ,V ;!
120010    003777  003777  170017  121442     !AKG AKG 8PO ZDB!!      P"#!
120020    121172  000000  000000  000000     !Y94              !!z"      !
120030    000000  000000  000000  000000     !                 !!        !
120040    000000  000000  054702  000000     !         NNR     !!     BY !
120050    000000  000000  000000  000000     !                 !!        !
120060    000000  000000  000000  000000     !                 !!        !
120070    000000  054744  000006  046166     !    NOL   F LIN!!   dY  vL!
120100    000000  046166  000000  046166     !    LIN     LIN!!   vL  vL!
120110    000000  046166  000000  046072     !    LIN     LG4!!   vL  :L!
120120    000000  046116  000000  000001     !    LHN       A!!   NL    !
120130    115274  120000  123077  000000     !XOL YX    ZW9   !!<    ?&  !
120140    000031  000000  000612  014006     ! Y        I4 C30!!         !
120150    004070  000001  000000  135600     !ALX   A      O  !!8       ;!
120160    053254  000000  100100  000000     !M4.        TT2  !!,V  @   !
120170    000000  000000  000000  000000     !                !!        !

            [ABOVE LINE REPEATED 60. TIMES]

121140    000000  006003  010400  000005     !    A65 B.2   E!!         !
121150    000001  000000  000000  054104     ! A           ND6!!      DX!
121160    000120  000000  000000  000000     ! B            !!P        !
121170    000000  006412  047105  042524     !    BCR LT7 KC.!!    ENTE!
121200    020122  020101  042524  046522     !EF4 EFQ KC. LN4!!R A TERM!
121210    047111  046101  047040  046525     !LUA LHA LT  LN7!!INAL NUM!
121220    042502  035122  042440  052116     !KCJ IMJ KBP MSV!!BER: ENT!
121230    051105  040440  041440  046517     !MFU JP2 J/X LN1!!ER A COM!
121240    040515  042116  043040  051117     !JQ7 J7F KH2 MF1!!MAND FOR!
121250    046440  051103  052516  041115     !LMX MFS MY8 JXM!! MCRNUMB!
```

Example 10-12   CDA of Task TT2
(Sheet 1 of 4)

```
RSX-11M CRASH DUMP ANALYZER      V4.0        1-NOV-81   11:39          PAGE 10
TASK DUMP


121260    051105   044440   020123   047516    !MFU K.  EF5 L$V!!ER IS NO!
121270    020124   041517   040524   026114    !EF6 J01 JRD GCT!!T OCTAL,!
121300    052040   054522   040440   040507    !MRP NKZ JP2 JQ1!! TRY AGA!
121310    047111   052041   051105   044515    !LUA MRQ MFU K/E!!IN!TERMI!
121320    040516   020114   047516   020124    !JQ8 EF. L$V EF6!!NAL NOT !
121330    047506   047125   020504   047516    !L$N LUM EL6 L$V!!FOUND!NO!
121340    050040   047517   020114   053101    !L22 L$W EF. M2A!! POOL AV!
121350    044501   040514   046102   020505    !K.3 JQ6 LHB EL7!!AILABLE!!
121360    027511   020117   051105   047522    !GVY EF1 MFU L$Z!!I/O ERRO!
121370    020122   047117   041440   046517    !EF4 LUG J/X LN1!!R ON COM!
121400    040515   042116   044440   050116    !JQ7 J7F K.  L38!!MAND INP!
121410    052125   000063   000000   000000    !MS/ AK         !!UT3      !
121420    000000   052124   000000   054104    !    MS.        ND6!! TT DX!
121430    054100   044336   044212   000002    !ND2 KZN KXJ  B!!@X^H H   !
121440    000021   005046   005046   005046    !  Q AX8 AX8 AX8!!  & & &  !
121450    012746   000040   012746   000033    !CTF   2 CTF   $!!f   f    !
121460    012746   121172   005046   005046    !CTF Y94 AX8 AX8!!f z'& &  !
121470    005046   112716   000001   012746    !AX8 W80   A CTF!!& N   f  !
121500    000005   012746   011000   012746    !  E CTF B5H CTF!!  f   f  !
121510    006003   104377   005046   005046    !A65 U61 AX8 AX8!!    & &  !
121520    005046   005046   012746   000004    !AX8 AX8 CTF   D!!& & f    !
121530    012746   121412   005046   012746    !CTF ZCR AX8 CTF!!f  #& f  !
121540    121416   005046   112716   000001    !ZCV AX8 W80   A!! #& N    !
121550    012746   000005   012746   010400    !CTF   E CTF B.2!!f   f    !
121560    012746   006003   104377   016703    !CTF A65 U61 DOS!!f     C  !
121570    177626   012700   121412   005002    ! 50 CSH ZCR AXB!!  @  #   !
121600    004767   001004   005702   001402    !AW1 L6 A5J  SJ!!w    B    !
121610    000167   000436   104376   121676    ! B9  GF U60 ZG8!!w   ~ >#!
121620    012700   010046   011000   001422    !CSH BWN B5H  SZ!!@ &      !
121630    026760   177566   000004   001372    !GN  48  D  SB!!P-v   z   !
121640    116067   000006   177570   116067    !X91  F  5 X91!!7   x 7   !
121650    000007   177564   020167   177560    !  G  46 EG1  42!!  t w P  !
121660    003361   010067   177546   062767    !ADQ BW1  4V PLW!!Q 7 f we!
121670    000001   177530   000207   005767    !  A  4H  CO A6W!!  X   w  !
121700    177522   001402   000167   000430    !  4B  SJ  B9  G !!R    w   !
121710    016700   177520   016002   000002    !DOP   4  DSJ  B!!@ P      !
121720    010104   166704   177510   016005    !BXD 8A.  32 DSM!!D DmH    !
121730    000010   070504   060502   010267    !  H RD6 OVR BZ9!!  DQBa7 !
121740    177470   012767   177777   177454    !  3F CTW  80  3D!!8 w  ,  !
121750    104376   122000   012701   000124    !U60 ZIX CSI  BD!!~  $A T !
121760    004767   067362   103404   005267    !AW1 QOB UXD A.W!!w  rn  7 !
121770    177432   010067   177432   000207    ! 2Z BW1  2Z  CO!!  7      !
122000    005767   177420   001402   000167    !A6W  2P  SJ  B9!!w     w !
122010    000406   012700   000124   016701    ! FV CSH  BD DOQ!!  @ T A !
122020    177406   012721   000040   077003    ! 2F CSY   2 TFS!!  Q   ~!
122030    016767   177374   177370   062767    !D11  16  12 PLW!!w !~x~we!
122040    000004   177362   005046   005046    !  D  1Z AX8 AX8!!  r~& &  !
122050    005046   012746   000040   012746    !AX8 CTF   2 CTF!!& f   f  !
122060    000027   012746   121225   005046    !  W CTF Z U AX8!!  f  '&  !
122070    005046   005046   112716   000001    !AX8 AX8 W80   A!!& & N    !
122100    012746   000005   012746   011000    !CTF   E CTF B5H!!f   f    !
122110    012746   006003   104377   005067    !CTF A65 U61 AYO!!f      7 !
122120    177274   005067   177272   005046    ! OL AYO  OJ AX8!!<~7 :~&  !
122130    005046   005046   005046   012746    !AX8 AX8 AX8 CTF!!& & & f  !
122140    000120   016746   177260   005046    ! P D1N  O  AX8!!P f O~&  !
122150    016746   177242   005046   112716    !D1N /Z AX8 W80!!f '~& N   !
```

Example 10-12   CDA of Task TT2
(Sheet 2 of 4)

```
122160    000001  012746  000005  012746     !  A CTF    E CTF!!    f    f !
122170    010400  012746  006003  104377     !B.2 CTF A65 U61!!    f       !
122200    103002  000167  000272  105767     !UQZ  B9   DZ VO1!!   w : w !
122210    177204  002002  000167  000260     !  .6  YZ   B9   DP!! ~  w O !
122220    104376  122244  016701  177200     !U60 ZM. DOQ  .2!!~ $$A   ~!
122230    016761  177176  000002  004767     !D1Y  .O   B AW1!!Q ~~   w !
122240    077170  000207  012746  000463     !TIP  CO CTF   G$!!x~   f 3 !
122250    104377  005046  005046  005046     !U61 AX8 AX8 AX8!!  &  &  & !
122260    012746  000040  012746  000037     !CTF   2 CTF   1!!f    f    !
122270    012746  121254  005046  005046     !CTF ZAD AX8 AX8!!f ,"& & !
122300    005046  112716  000001  012746     !AX8 W80   A CTF!!& N    f !
122310    000005  012746  011000  012746     !  E CTF B5H CTF!!    f    f !
122320    006003  104377  012700  121412     !A65 U61 CSH ZCR!!    @   #!
122330    005020  077302  000167  177102     !AXP TKJ  B9  $J!!   B~w B~!
122340    005046  005046  005046  012746     !AX8 AX8 AX8 CTF!!& & & f !
122350    000040  012746  000023  012746     !  2 CTF   S CTF!!   f    f !
122360    121313  005046  005046  005046     !ZA5 AX8 AX8 AX8!!K"& & & !
122370    112716  000001  012746  000005     !W80   A CTF   E!!N    f    !
122400    012746  011000  012746  006003     !CTF B5H CTF A65!!f    f    !
122410    104377  012746  000463  104377     !U61 CTF   G$ U61!!   f 3   !
122420    005046  005046  005046  012746     !AX8 AX8 AX8 CTF!!& & & f !
122430    000040  012746  000022  012746     !  2 CTF   R CTF!!   f    f !
122440    121336  005046  005046  005046     !ZBN AX8 AX8 AX8!!~"& & !
122450    112716  000001  012746  000005     !W80   A CTF   E!!N    f    !
122460    012746  011000  012746  006003     !CTF B5H CTF A65!!f    f    !
122470    104377  012746  000463  104377     !U61 CTF   G$ U61!!   f 3   !
122500    005046  005046  005046  012746     !AX8 AX8 AX8 CTF!!& & & f !
122510    000040  012746  000032  012746     !  2 CTF   Z CTF!!   f    f !
122520    121360  005046  005046  005046     !ZB2 AX8 AX8 AX8!!P"& & & !
122530    112716  000001  012746  000005     !W80   A CTF   E!!N    f    !
122540    012746  011000  012746  006003     !CTF B5H CTF A65!!f    f    !
122550    104377  104376  122574  016700     !U61 U60 ZSD DOP!!  ~ !%@ !
122560    176646  012701  000124  004767     !  WN CSI   BD AW1!!&}A T w !
122570    067022  000207  012746  000463     !QXR  CO CTF   G$!!  n  f 3 !
122600    104377  012702  000012  000402     !U61 CSJ   J   FR!!  B    !
122610    012702  000010  004567  000144     !CSJ   H ATW  BT!!B    w d !
122620    005001  112005  122705  000040     !AXA W$M ZT7   2!!    E% !
122630    001774  122705  000011  001771     ! YT ZT7   I  YQ!!! E% y !
122640    162705  000060  120502  103010     !60U  AH Y2B UQ2!!EeO B!  !
122650    010004  010200  004767  000016     !BVT BYX AW1   N!!    w    !
122660    010400  060501  112005  000764     !B.2 OVQ W$M  LT!!  Aa  t !
122670    116002  177777  000207  010046     !X8R  80  CO BWN!!      & !
122700    012746  000021  005000  006000     !CTF   Q AX A62!!f        !
122710    006001  103002  066600  000002     !A63 UQZ QT2   B!!    m   !
122720    005316  003371  000415  012746     !A/F ADY  F/ CTF!!N y    f !
122730    000020  010146  005001  006300     !  F BX8 AXA BAX!!  f    @ !
122740    006101  020116  103402  161601     !A8Q EFO UXB 6PA!!A N    c!
122750    005200  005366  000002  003367     !A$H AOF   B ADW!!  v    w !
122760    022626  000207  010446  010346     !F V  CO B/O B.F!! %  & f !
122770    010546  016605  000006  004736     !B1N D.7   F AWF!!f       ^ !
123000    012603  012604  012605  000207     !CQ$ CQ. CQ/  CO!!          !
123010    000000  000000  000000  000000     !                !!        !
123020    000000  000000  000000  000000     !                !!        !
123030    000000  000000  000000  000000     !                !!        !
123040    000000  000000  000000  000000     !                !!        !
123050    000000  000000  000000  000000     !                !!        !
```

Example 10-12   CDA of Task TT2
(Sheet 3 of 4)

RSX-11M CRASH DUMP ANALYZER   V4.0   1-NOV-81   11:39   PAGE 12
TASK DUMP

```
123060   000000  000000  000000  000000    !              !!          !
123070   000000  000000  000000  000000    !              !!          !
```

[END OF ANALYSIS OUTPUT]

Example 10-12 CDA of Task TT2
(Sheet 4 of 4)

DIRECTIVE PROCESSING

11

INTRODUCTION

Many tasks require activities, such as I/O, which cannot be easily performed within a user task. To provide the facilities for these activities without requiring privileged tasks and extensive knowledge of the inner workings of the operating system, RSX-11M provides a set of system services called Executive directives.

Executive directives provide a simple interface to these services. This interface is through system macros for the MACRO-11 programmer and system subroutines for the FORTRAN programmer. The interface also provides a completion code for error checking.

OBJECTIVES

1. List the steps in processing an Executive directive.

2. Perform the steps necessary to add a user written directive to the system.

RESOURCES

1. RSX-11M System Lists and Data Structures

2. RSX-11M/M-PLUS Executive Reference Manual

THE USER TASK INTERFACE

- MACRO-11 task uses macros to invoke directives

- Three types of macros

        $-form
        $C-form
        $S-form

- FORTRAN tasks call subroutines

- Directive Parameter Block (DPB) used to pass data to system

- Result of directive execution indicated by

    Directive Status Word (DSW)

    Processor Status Word
        Set to indicate failure
        Clear to indicate success

Directive Parameter Block

- Data structure contained within the user task

- Contains parameters needed to execute the directive

    Directive identification code (DIC)
    Other parameters as needed
    Length depends on DIC, can be variable

- Location of DPB

    In task data area

        In user selected PSECT
        In PSECT $DPB$$

    On task stack

- Directive identification code always odd whole number
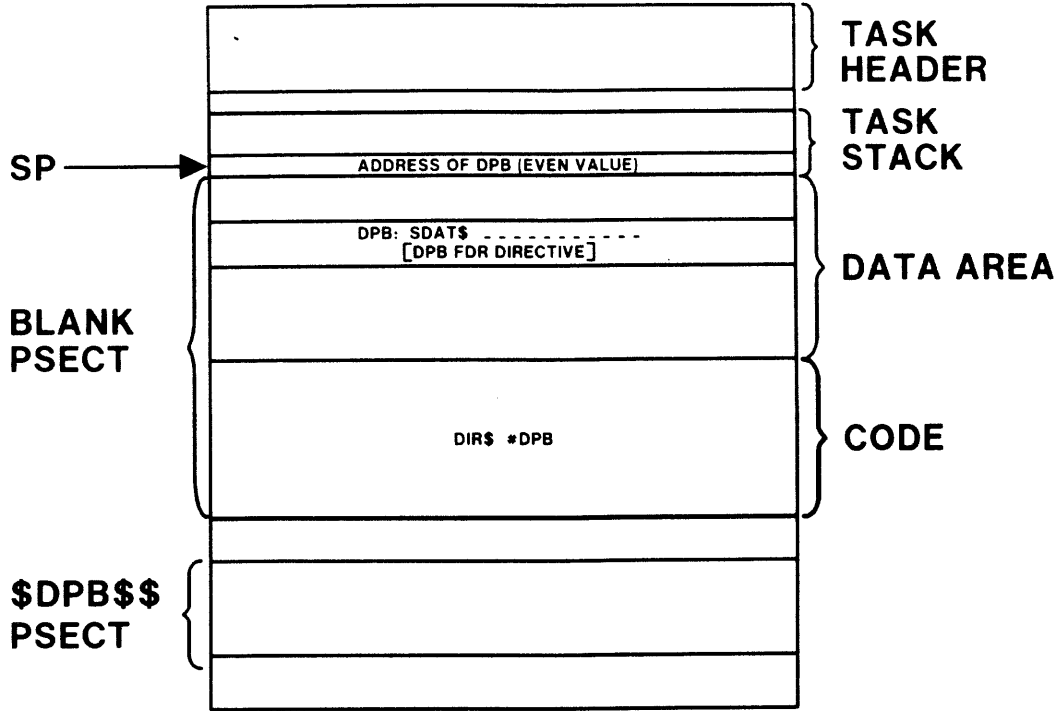
```
                                                    ┐ TASK
                                                    ┘ HEADER

                                                    ┐ TASK
                                                    │ STACK
SP ──────►  ADDRESS OF DPB (EVEN VALUE)             ┘

            DPB: SDAT$ - - - - - - - - - -          ┐
                  [DPB FOR DIRECTIVE]               │ DATA AREA
BLANK                                               ┘
PSECT  {
                                                    ┐
                                                    │ CODE
              DIR$  #DPB                            │
                                                    ┘

$DPB$$ {
PSECT
```
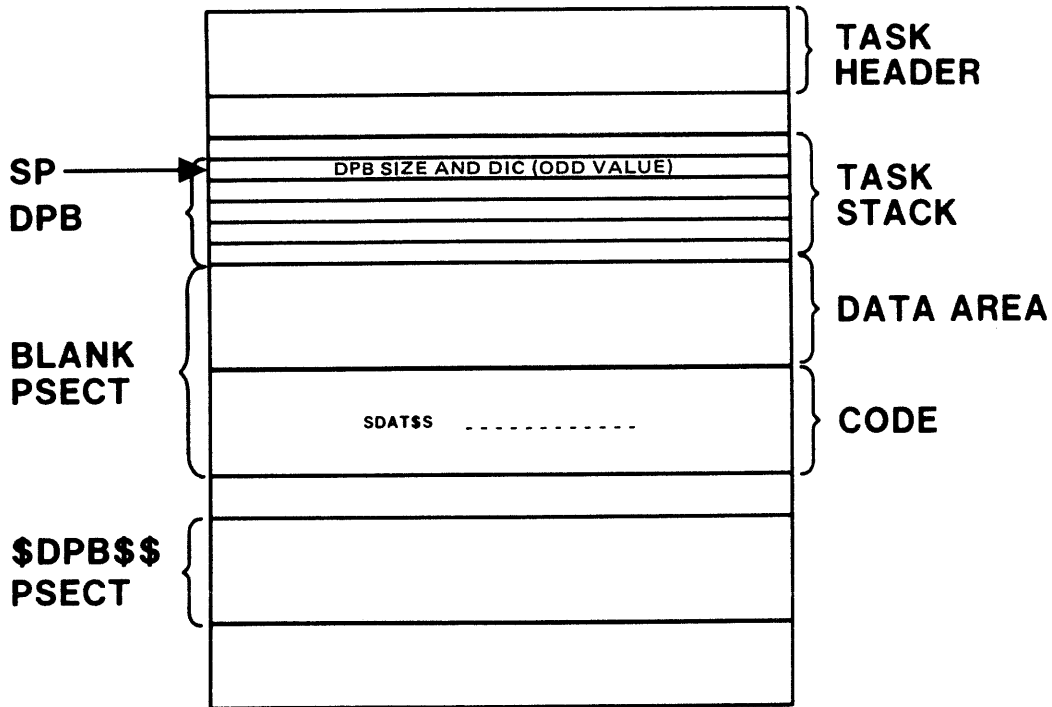
Figure 11-1   Directive Macros: '$' Form

```
                                                    ┐ TASK
                                                    ┘ HEADER

SP ──────►  DPB SIZE AND DIC (ODD VALUE)            ┐ TASK
DPB   {                                             ┘ STACK

                                                    ┐ DATA AREA
                                                    ┘
BLANK
PSECT {
               SDAT$S  - - - - - - - - - -          ┐ CODE
                                                    ┘

$DPB$$ {
PSECT
```

Figure 11-1   Directive Macros: '$S' Form

Figure 11-1 Directive Macros: '$C' Form

'$' FORM

```
SEND:   SDAT$   TASKB,BUF,5           SEND:        .BYTE   71.,5
                                                   .RAD50  /TASKB/
                                                   .WORD   BUF
                                                   .WORD   5
        DIR$    #SEND,ERR                          MOV     #SEND,-(SP)
                                                   EMT     377
                                                   BCC     .+6
                                                   JSR     PC,ERR
```
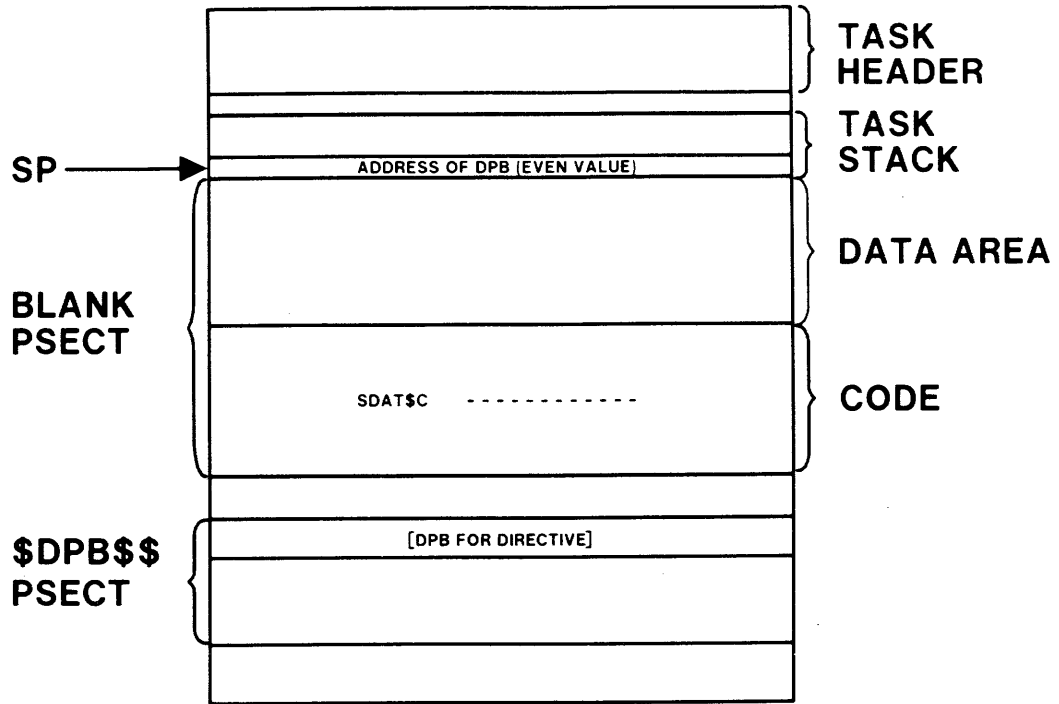

'$C' FORM

```
                                                   .PSECT  $DPB$$
                                      $$$          =       .
                                                   .BYTE   71.,5
                                                   .RAD50  /TASKB/
        SDAT$C  TASKB,BUF,5,PROG1,ERR              .WORD   BUF
                                                   .WORD   5
                                                   .PSECT  PROG1
                                                   MOV     #$$$,-(SP)
                                                   EMT     377
                                                   BCC     .+6
                                                   JSR     PC,ERR
```

'$S' FORM

```
TASKB:  .RAD50  /TASKB/               TASKB:       .RAD50  /TASKB/

                                                   MOV     #5,-(SP)
                                                   MOV     #BUF,-(SP)
                                                   MOV     TASKB+2,-(SP)
        SDAT$S  #TASKB,#BUF,#5,ERR                 MOV     TASKB,-(SP)
                                                   MOV     (PC)+,-(SP)
                                                   .BYTE   71.,5
                                                   EMT     377
                                                   BCC     .+6
                                                   JSR     PC,ERR
```

Example 11-1   Macro Expansions for SEND DATA

DPB SIZE IN WORDS	DIC
PARAMETER 1	
PARAMETER 2	
PARAMETER N	

Figure 11-2 The Directive Parameter Block

Directive Status Word

- Executive returns a status code in the DSW

- In general, success is denoted by a status of +1

- The Set/Clear Event Flag directives use

    0 to indicate success and flag clear

    2 to indicate success and flag set

- An error is indicated by a negative status

- Value stored at location $DSW in the task header (in task region)

- In FORTRAN, value returned in calling argument

THE DIRECTIVE DISPATCHER

- EMT trap causes entry to module DRSUB at $EMTRP

- Module DRDSP is mapped using KERNEL APR5 and control passes to module DRDSP at $DRDSP

- Routine $DRDSP distinguishs between directive calls, calls to $SWSTK and non-RSX EMT traps

- For directive calls

  KERNEL APR6 is used to map

  WDB or RDB if PLAS directive
  DPB otherwise

  Directive processing routine is called

- To process a request to switch to kernel mode, a jump to the Executive routine $SWSTK is executed

- For non-RSX EMT traps, a jump to $EMSST in SSTSR is executed

The Directive Dispatch Table

- Contains entry for each directive

  Address of service routine
  'Flags byte'
  Length in words (For variable length = Ø)

- Assembled in one of two possible forms

  Single table form

  Two words per possible directive

  Double table form

  One byte per possible directive
  Two words per existing directive

- The form which uses the least space is automatically chosen

  If less than 75 percent of directives exist, use the two table structure

  If 75 percent or more of the possible directives exist, use the one table structure

297

QIO$	1.	ADDRESS OF $DRQIO		2.
		FLAGS BYTE	DPB LENGTH	4.
QIOW$	3.	ADDRESS OF $DRQIO		6.
		FLAGS BYTE	DPB LENGTH	8.
GLUN$	5.	ADDRESS OF $DRGLI		10.
		FLAGS BYTE	DPB LENGTH	12.
ALUN$	7.	ADDRESS OF $DRASG		14.
		FLAGS BYTE	DPB LENGTH	16.
ALTP$	9.	0		18.
		0	0	20.
RQST$	11.	ADDRESS OF $DRREQ		22.
		FLAGS BYTE	DPB LENGTH	24.
UNUSED	13.	0		26.
		0	0	28.
UNUSED	15.	0		30.
		0	0	32.
RUN$	17.	ADDRESS OF $DRRUN		34.
		FLAGS BYTE	DPB LENGTH	36.
UNUSED	19.	0		40.
		0	0	42.
SRRA$	21.	ADDRESS OF $DRRRA		44.
		FLAGS BYTE	DPB LENGTH	46.
MRKT$	23.	ADDRESS OF $DRMKT		50.
		FLAGS BYTE	DPB LENGTH	52.

Single Table Form

TABLE 1   TABLE 2

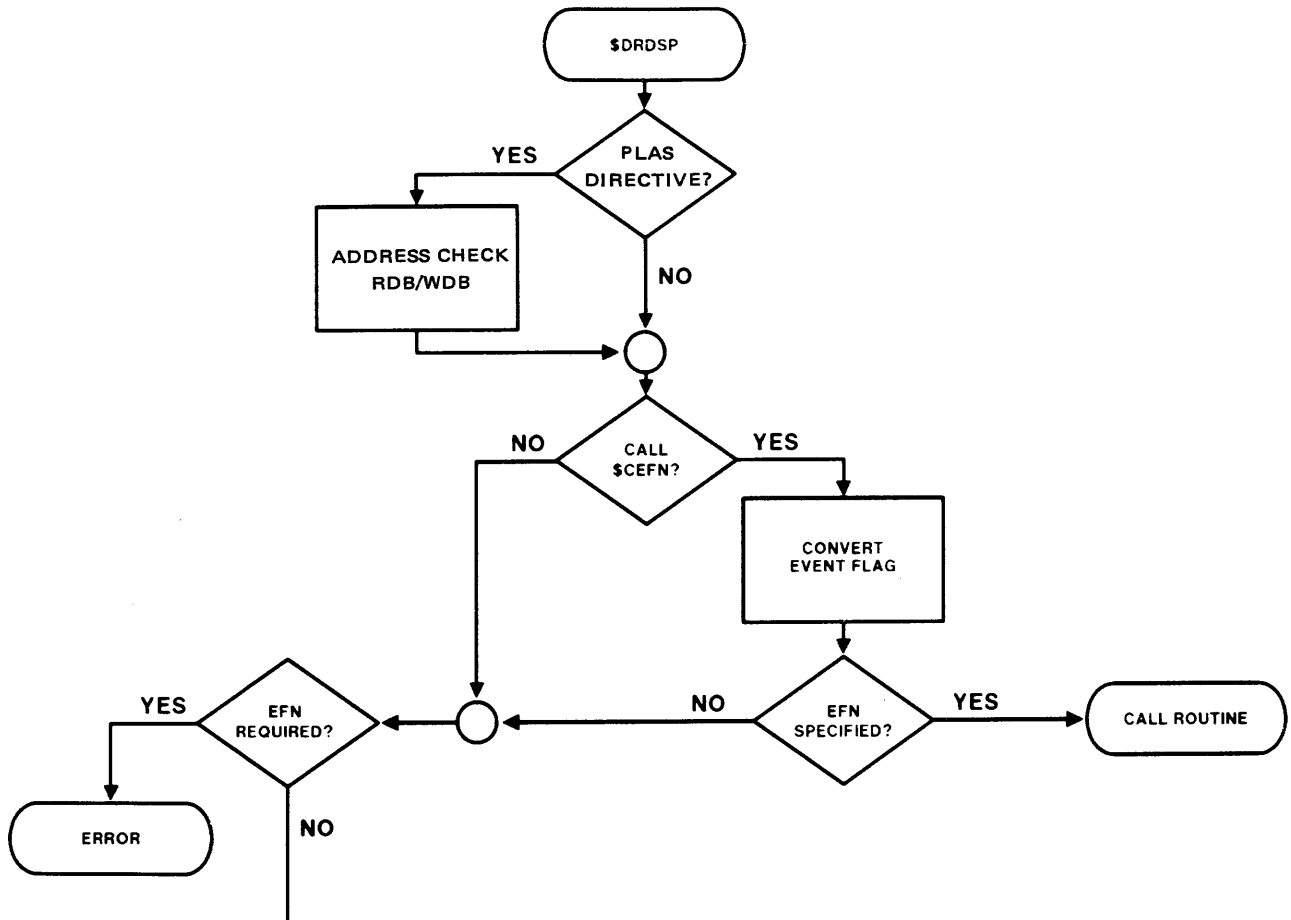|						
QIO$	1.	2.		ADDRESS OF $DRQIO		2.
QIOW$	3.	6.		FLAGS BYTE	DPB LENGTH=12.	4.
GLUN$	5.	10.		ADDRESS OF $DRQIO		6.
ALUN$	7.	14.		FLAGS BYTE	DPB LENGTH=12.	8.
ALTP$	9.	0		ADDRESS OF $DRGLI		10.
RQST$	11.	18.		FLAGS BYTE	DPB LENGTH=3.	12.
UNUSED	13.	0		ADDRESS OF $DRASG		14.
UNUSED	15.	0		FLAGS BYTE	DPB LENGTH=4.	16.
RUN$	17.	22.		ADDRESS OF $DRREQ		18.
UNUSED	19.	0		FLAGS BYTE	DPB LENGTH=7.	20.
SRRA$	21.	26.		ADDRESS OF $DRRUN		22.
MRKT$	23.	30.		FLAGS BYTE	DPB LENGTH=11.	24.
CSRQ$	25.	34.		ADDRESS OF $DRRRA		26.
CMKT$	27.	38.		FLAGS BYTE	DPB LENGTH=2.	28.
EXST$	29.	42.		ADDRESS OF $DRMKT		30.
CLEF$	31.	46.		FLAGS BYTE	DPB LENGTH=5.	32.
SETF$	33.	50.				

Double Table Form

Figure 11-3   The Directive Dispatch Table
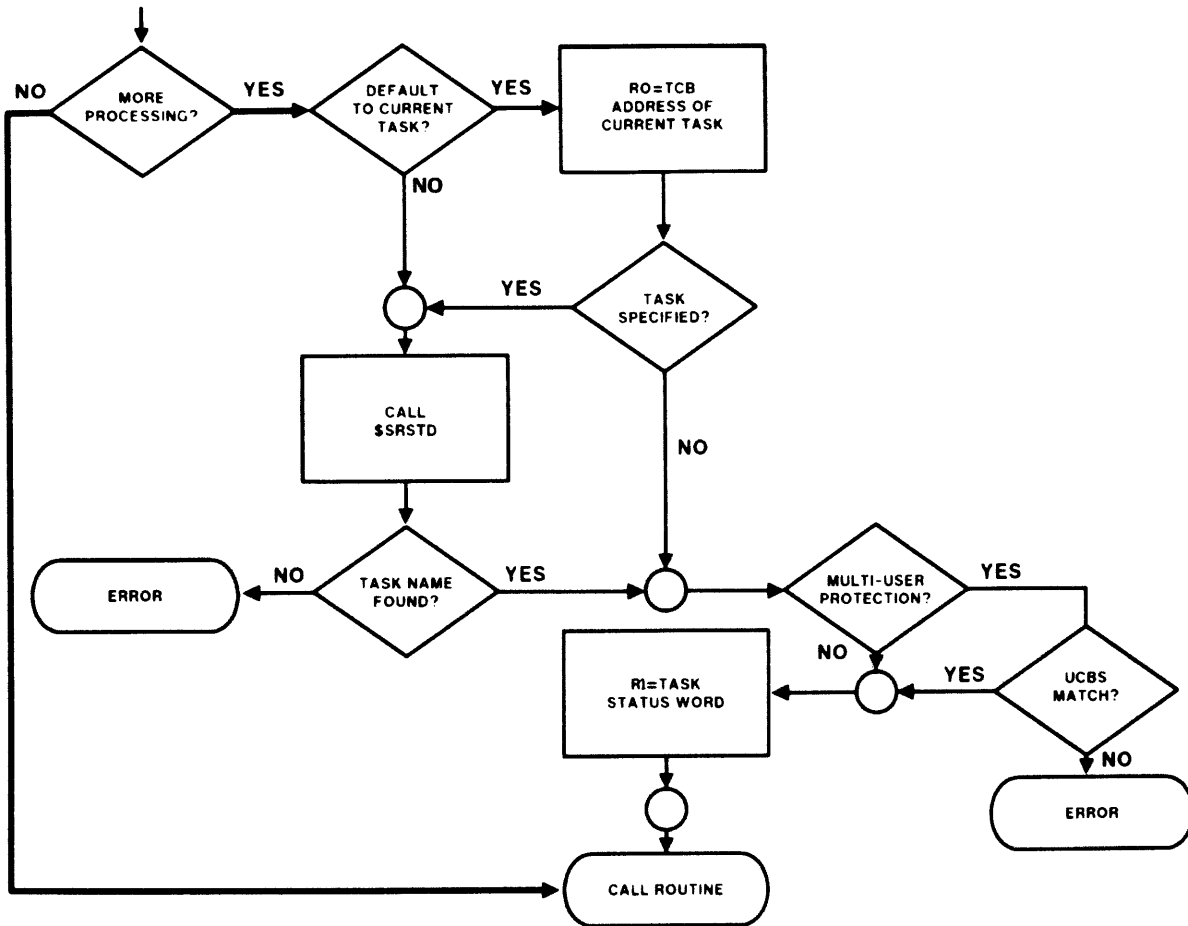
Figure 11-4   Processing in DRDSP
(Sheet 1 of 2)

Figure 11-4 Processing in DRDSP
(Sheet 2 of 2)

DIRECTIVE PROCESSING ROUTINES

The Interface

- Directive processing routine is called with registers loaded as in Table 11-1

- Flag byte in directive dispatch table determines processing

  ACHKDB – Address PLAS definition block

  CEFNCL – Convert the event flag

  GEFUSE – Do not increment group global use count

  CEFNMT – Event flag must be specified

  DFCTSK – Task name defaults to current task

  MUPCHK – Perform multi-user protection check

  SRSTCL – Call $SRSTD to find TCB in STD

- Bits set in flag byte determine contents of RØ and R1

  If CEFNCL specified
      RØ – Event flag mask word
      R1 – Address of event flag

  If SRSTCL specified
      RØ – TCB address of specified task
      R1 – Address of status word in TCB

Table 11-1   Register Usage

Register	Contents
R5	Address of the TCB of the current task
R4	Address of the header of the current task
R3	Address of the next word in the DPB
R2	Address of the second task status word of the current task
R1	Depends on directive
R0	Depends on directive

The Directive Commons

- Most directive processing code removed from Executive

    Increases available DSR

- Two directive commons, EXCOM1 and EXCOM2

    Existence of commons determined at SYSGEN

    Kernel I-space APR 5 used to map directive commons

    Location of processing routine indicated by low order bit of address of routine in DDT

- Module DRSUB

    Is in the Executive

    Always mapped

    Contains the location $EMTRP

        Calls $DIRSV
        Maps to DRDSP which is in EXCOM1
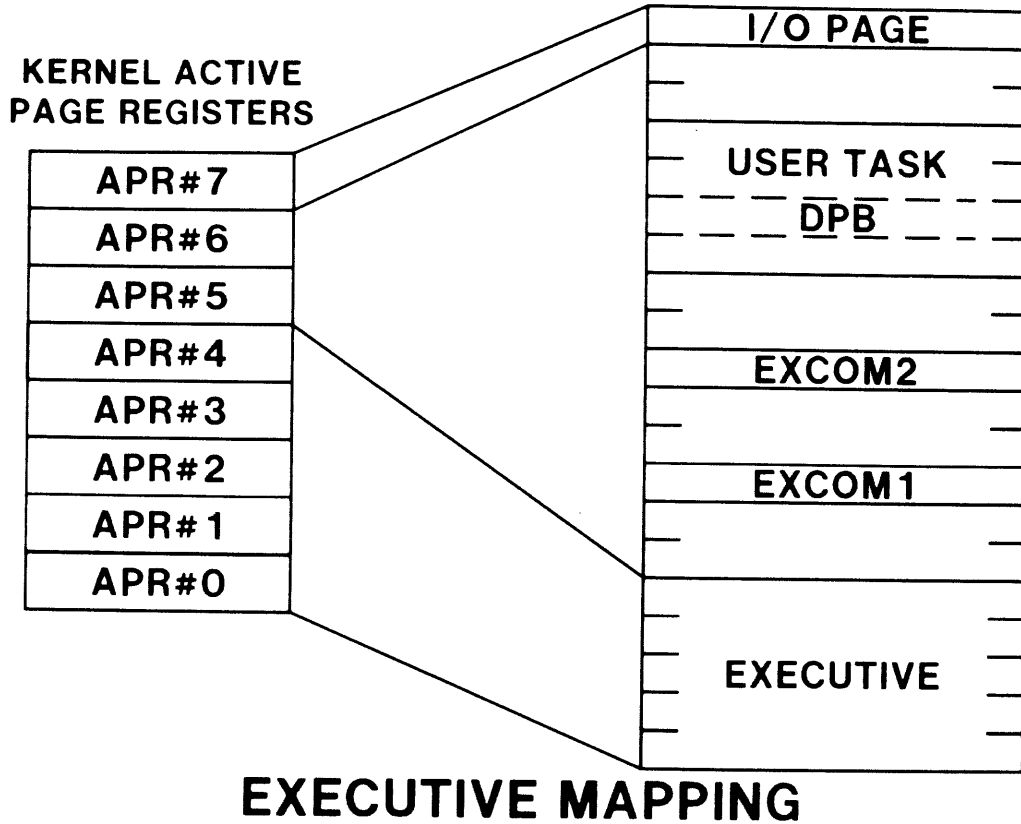   Contains coroutines $MPXC1 and $MPXC2 which are used to map EXCOM1 and EXCOM2

KERNEL ACTIVE PAGE REGISTERS

APR#7
APR#6
APR#5
APR#4
APR#3
APR#2
APR#1
APR#0

I/O PAGE

USER TASK

DPB

EXCOM2

EXCOM1

EXECUTIVE

EXECUTIVE MAPPING

Figure 11-5   The Directive Commons: Standard Executive Mapping

KERNEL ACTIVE PAGE REGISTERS

I/O PAGE
USER TASK
DPB
EXCOM2
EXCOM1
EXECUTIVE

APR#7
APR#6
APR#5
APR#4
APR#3
APR#2
APR#1
APR#0

Figure 11-5 The Directive Commons: Mapping to Reference EXCOM1

Table 11-2 Directive Processing Modules

Module	Location	Routines
DRABO	EXCOM1	$DRABO
DRASG	EXCOM2	$DRATX
DRATP	EXCOM2	
DRATX	EXCOM1	
DRCIN	EXCOM2	$DRCIN, $DISIN
DRCLI	EXCOM1	$DRCLI, $STCLI, $STCL1
DRCMT	EXCOM1	
DRDAR	EXCOM1	$DRDAR, $DREAR
DRDCP	EXEC	$DRDCP, $DRECP
DRDSP	EXCOM1	$TRTRP, $DRTRP, $EMTRP, $DRDSP, $DRATP
DREIF	EXEC	$DREIF, $DREXT, $DREX1
DREXP	EXCOM2	$DREXP
DRGCL	EXCOM1	$GRGCL, $RLMCB, $RLCIB
DRGEF	EXCOM2	$DRCRE, $DRELE, $UNLCK, $GTGEF
DRGLI	EXCOM1	$DRGLI
DRGPP	EXCOM1	$DRGPP
DRGSS	EXCOM1	$DRGSS
DRGTK	EXCOM1	$DRGTK
DRGTP	EXCOM2	$DRGTP
DRMAP	EXCOM2	$DRCRW, $DRELW, $DRMAP, $DRUNM, $DRSRF, $DRRRF $DRGMX
DRMKT	EXCOM1	$DRMKT, $DRRUN
DRPUT	EXCOM1	$DRREX, $CAAST, $DRFEX, $DRPUT, $DRRRA, $DRRCV
DRQIO	EXCOM1	$DRQIO
DRRAS	EXCOM1	$DRRCS, $DRREC, $DRSND
DRREG	EXCOM2	
DRRES	EXCOM1	$DRUNS, $DRRES, $DRSTP, $DRSPN
DRSED	EXCOM1	$DRCEF, $DRRAF, $DRSEF, $DRSTS, $DRSTL, $DRWFL $DRWFS
DRSMG	EXCOM2	
DRSPW	EXCOM1	
DRSST	EXCOM1	$DRSDV, $DRSTV
DRSUB	EXEC	$ELGEF, $DEAGF, $SRGEF, $DRDSE, $TKWSE, $DRWSE $MPXC1, $MPXC2, $DETRG, $CITFR, $DRQRQ, $SRCCQ $GTUCB, $STCLI, $FNCLI, $TRTRP, $EMTRP, $OTHR1 $DRATP

Send/Receive Processing

● Used to transfer a 13-word data packet to a specified task

● Routine $DRSND in module DRRAS used to send data

● Routine $DRREC in module DRRAS used to receive data

● Steps in sending/receiving

An 18-word send/receive data block is allocated in DSR

Link word for receive queue
Sender name in RAD50
13-word data area
UCB of TI: of sender
Current UIC of sender

A 13-word buffer in sender task is copied into DSR block

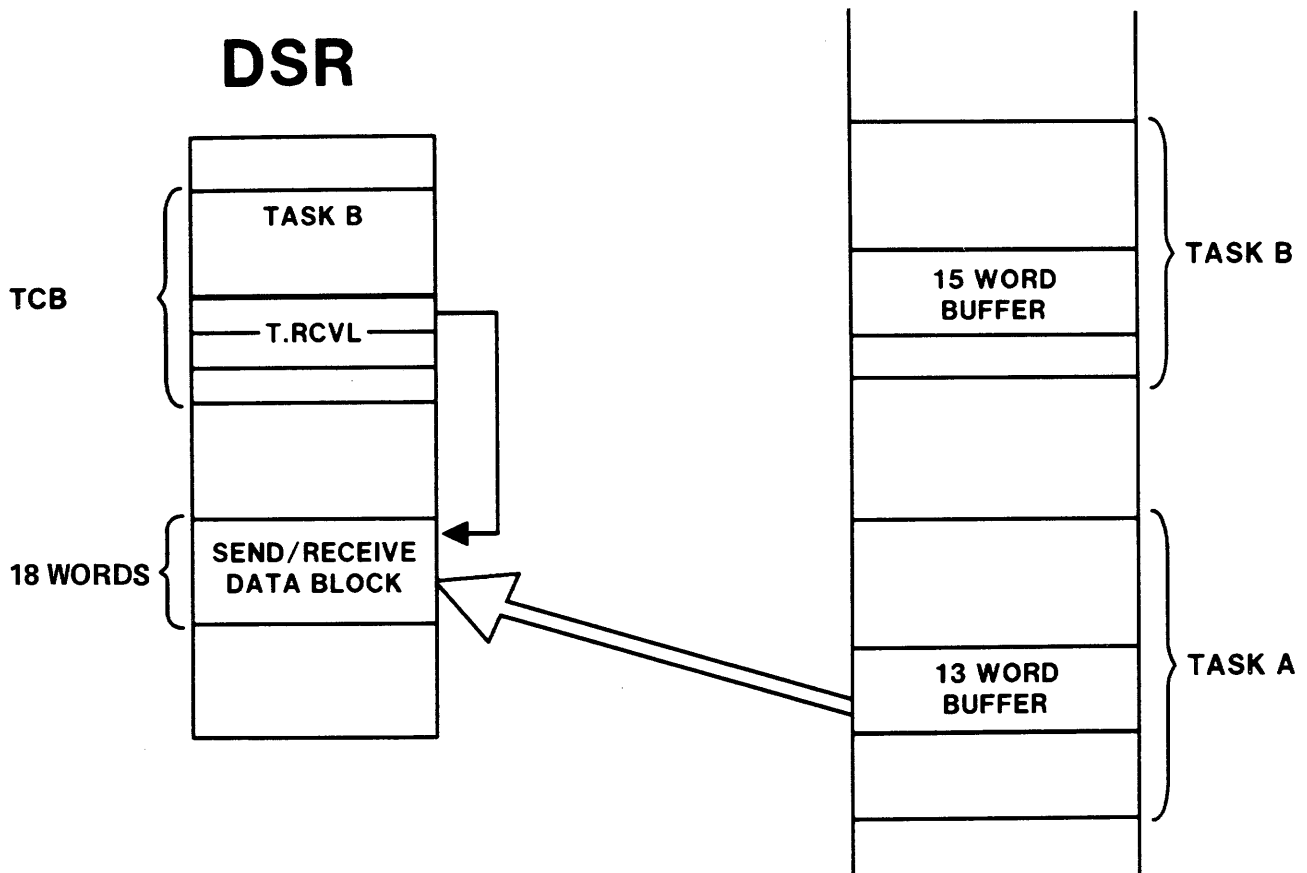A 15-word buffer in user task receives sender name and data

Figure 11-6   Send/Receive Processing: Send from TASK A

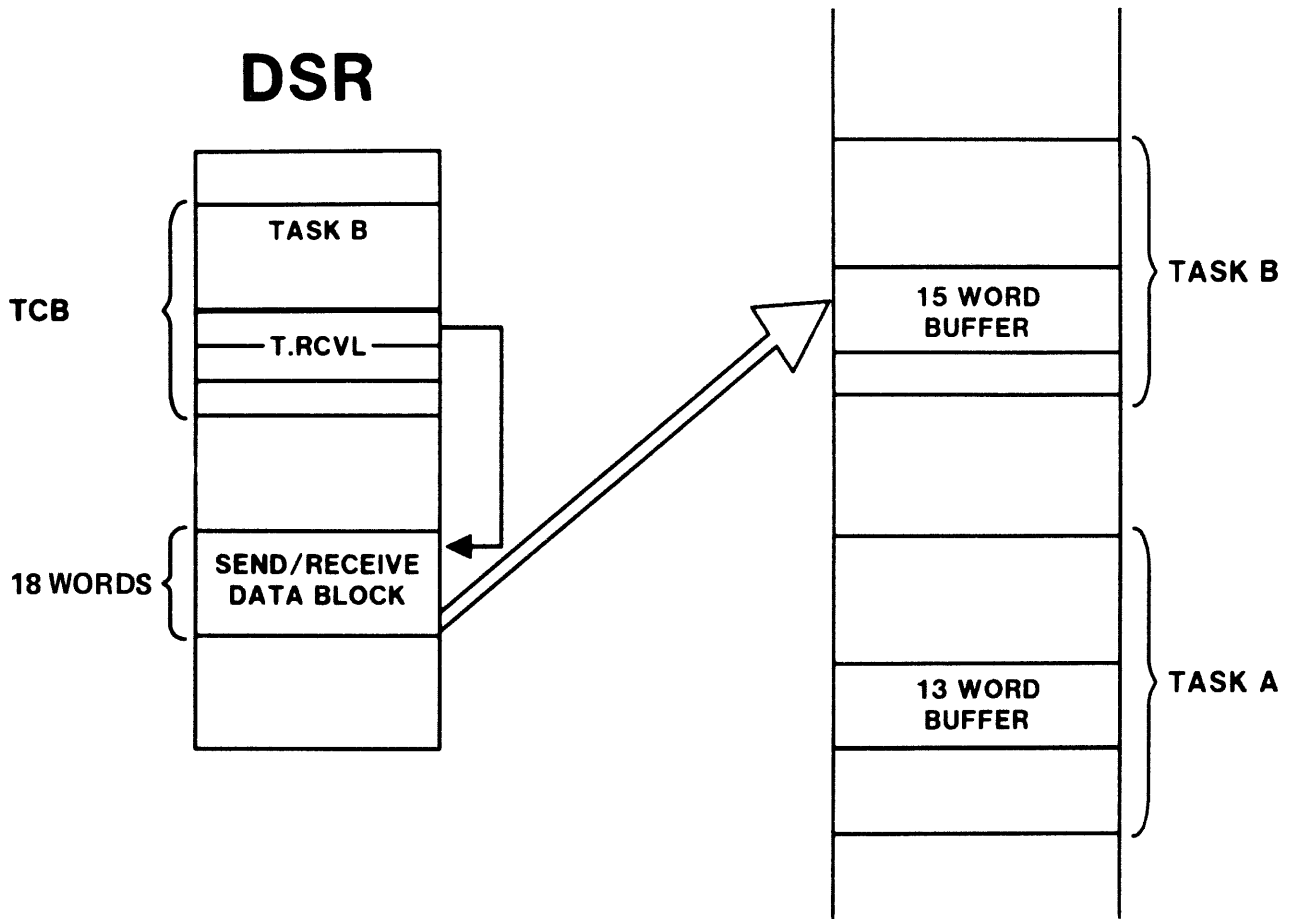Figure 11-6 Send/Receive Processing: Receive by TASK B

Parent/Offspring Tasking

- Connecting TASKA to TASKB

    TASKB passes a one-word status back to TASKA when

        TASKB exits
        TASKB emits status

    TASKA is the parent task, TASKB is the offspring task

- TASKA spawns TASKB

    TASKA requests TASKB

    TASKA connects to TASKB

- Steps in connecting

    A 14-word offspring control block is allocated in DSR

    OCB is linked to the TCB of the offspring task at T.OCBH

    When offspring exits or emits status, status moved to OCB
    placed into first word in 8-word status area

    Status automatically moved into parent status buffer

    Parent task specifies event flag and/or AST for
    notification of the offspring's return of status

OCB LINK WORD	0	O.LINK
ADDRESS OF MCR COMMAND LINE	2	O.MCRL
PARENT TCB ADDRESS	4	O.PTCB
EXIT AST ADDRESS	6	O.AST
EXIT EVENT FLAG	10	O.EFN
EXIT STATUS BLOCK VIRTUAL ADDRESS	12	O.ESB
	14	O.STAT

EXIT STATUS
BUFFER
(8 WORDS)

16
20
22
24
26
30
32

Figure 11-7   Parent/Offspring Tasking: Offspring Control Block
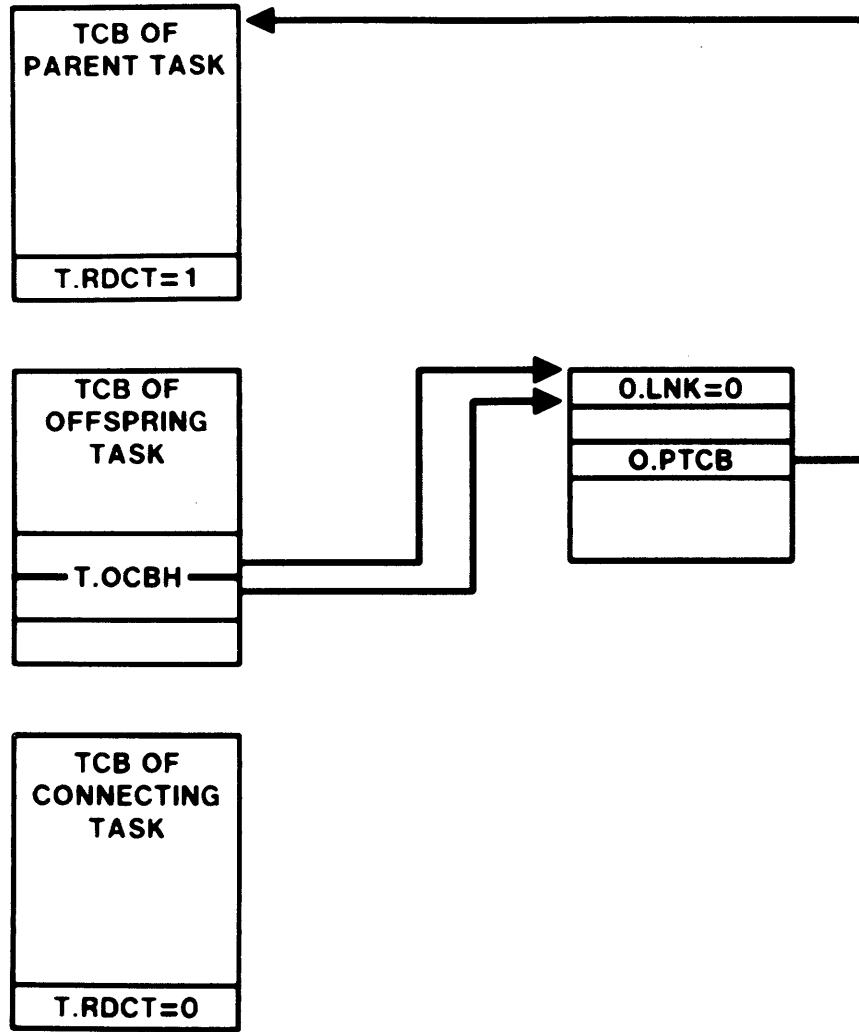
Figure 11-7   Parent/Offspring Tasking: Spawning a Task

Figure 11-7   Parent/Offspring Tasking: Connecting to a Task

Memory Management Processing

- User task specifies information for creating/attaching to a region in a Region Definition Block (RDB)

- Information in RDB used to initialize PCB for subpartition in dynamic partition

- User task specifies information for creating/mapping an addressing window in a Window Definition Block (WDB)

- Information in WDB used to initialize window block in task header

REGION DEFINITION BLOCK

REGION ID
SIZE OF REGION (32W BLOCKS)
NAME OF REGION (RAD50)
REGION'S MAIN PARTITION NAME (RAD50)
REGION STATUS WORD
REGION PROTECTION WORD

PARTITION CONTROL BLOCK (PCB)

LINK TO NEXT PARTITION PCB	
I/O COUNT=0	PRIORITY OF PARTITION
PARTITION NAME (IN RADIX-50)	
POINTER TO NEXT SUB-PARTITION	
POINTER TO MAIN	
PHYSICAL START ADDRESS OF PARTITION	
SIZE OF PARTITION	
PARTITION WAIT QUEUE LIST HEAD	
PARTITION SWAP SIZE (SYSTEM ONLY)	
PARTITION BUSY FLAGS	
TCS ADDR. OF OWNER/CURRENT UIC OF OWNER TASK	
PARTITION STATUS FLAGS	
POINTER TO TASK HEADER	
PROTECTION WORD FOR REGION	
ATTACHMENT DESCRIPTOR LIST HEAD	

CREATING A REGION

WINDOW DEFINITION BLOCK

BASE APR	WINDOW ID
VIRTUAL BASE ADDRESS (BYTES)	
WINDOW SIZE (32W BLOCKS)	
REGION ID	
OFFSET IN REGION (32W BLOCKS)	
LENGTH TO MAP (32W BLOCKS)	
WINDOW STATUS WORD	
SEND/RECEIVE BUFFER ADDRESS (BYTES)	

TASK ADDRESSING WINDOW BLOCK

PCB ADDRESS	
LOW VIRTUAL ADDRESS LIMIT	
HIGH VIRTUAL ADDRESS LIMIT	
ADDRESS OF ATTACHMENT DESCRIPTOR	
SIZE OF WINDOW IN 32. WORD BLOCKS	
PHYSICAL MEMORY OFFSET IN 32. WORD BLOCKS	
NUMBER OF APRS TO MAP	FIRST PDR ADDRESS (LOW BYTE)
CONTENTS OF LAST PDR	

CREATING AND MAPPING A WINDOW

Figure 11-8   Memory Management Processing

314

ADDING A USER-WRITTEN DIRECTIVE

● Interface to the directive processing

Structure of Directive Parameter Block must be determined

Entry for new directive must be added to the DDT

Directive processing routine must be written

Macros to invoke directive must be written

Directive processing routine must be built into the system

Macros must be added to the system macro library

Structuring the Directive Parameter Block

● Information needed to execute directive

● Choose an unused DIC

● Choose order for necessary parameters

If event flag is first parameter, specify CEFNCL and/or CEFNMT

If task name is first parameter, specify DFCTSK and/or SRSTCL

SIZE=4	DIC=161
TASK NAME	
IN RAD50	
EVENT FLAG NUMBER	

Figure 11-9 Directive Parameter Block for New Directive

Changes to the Directive Dispatcher

- DDT determines

    Which directive processing routine to call

    What processing to perform in DRDSP

    Size of DPB

    Where the module is to be located (directive commons)

- Each directive is represented by a line in the DIREC macro defined in the module DRDSP

- Format of line in DIREC macro is

        MAC   DIC,DPBSIZ,DIRADR,MASK,COND,SECOND

    Where

        DIC   -   Directive Identification Code
        DPBSIZ   -   Size of the DPB
        DIRADR   -   Address of the Directive Processing Routine
        MASK   -   Directive Processing Control Mask
        COND   -   Conditional Assembly Parameters which must
                      be defined if the directive is supported
        SECOND   -   Conditional assembly parameters which if
                      defined require that the directive have a
                      variable length DPB

    Example:

        MAC   55.,2.,$DRCRR!PAG,<ACHKDB>,P$$LAS

    This is the 'create region' directive.

    PAG is set to 1 if directive common support is
    desired, 0 otherwise. The service routine will be
    placed in EXCOM2.

Place the following in the Directive Dispatch Table

          MAC   161.,4.,$DRFLG!PAG,<DFCTSK!SRSTCL>

DIC
Size of DPB
Processing Routine
Force to EXCOM2
Default to current task
Call $SRSTD to find TCB

Example 11-2 New Entry in the Directive Dispatch Table

The Directive Processing Routine and Macros

- Routine should be written to make use of the standard register usage in DRDSP

- Assembly with EXEMC and RSXMC

- Provide only macros needed by applications

```
        .TITLE  DRFLG
        .IDENT  /1.0/
;+
;  ** - $DRFLG - SET TASK'S LOCAL EVENT FLAG
;
;  THIS DIRECTIVE INSTRUCTS THE SYSTEM TO SET A SPECIFIED TASK'S LOCAL
;  EVENT FLAG.
;
;  DPB FORMAT:
;
;       WD. 00 -- DIC (161.), DPB SIZE (4.)
;       WD. 01 -- FIRST HALF OF TASK NAME (RAD50)
;       WD. 02 -- SECOND HALF OF TASK NAME (RAD50)
;       WD. 03 -- EVENT FLAG NUMBER (1-32.)
;
;  INPUTS:
;       R0 = TCB ADDR. OF TARGET TASK
;       R1 = ADDR. OF FIRST TASK STATUS WORD
;       R2 = ADDR. OF SECOND TASK STATUS WORD
;       R3 = ADDR. OF NEXT WORD IN DPB
;       R4 = ADDR. OF HEADER OF CURRENT TASK
;       R5 = ADDR. OF TCB OF CURRENT TASK
;
;  OUTPUTS:
;       C = 0 IF DIRECTIVE IS SUCCESSFULLY COMPLETED
;               DIRECTIVE STATUS D.RS22 IS RETURNED IF EVENT FLAG IS
;                   ALREADY SET
;               DIRECTIVE STATUS D.RS00 IS RETURNED IF EVENT FLAG WAS
;                   CLEARED
;       C = 1 IF DIRECTIVE IS REJECTED
;               DIRECTIVE STATUS D.RS97 IS RETURNED IF ILLEGAL EVENT
;                   FLAG
;-
        .ENABL  LSB
$DRFLG::
        MOV     R0,R5      ;ADDR OF TARGET TASK TCB
        CALL    $CEFN      ;CONVERT TO EVENT FLAG MASK
        BCS     20$        ;ILLEGAL EVENT FLAG
        BIT     R0,(R1)    ;EVENT FLAG ALREADY SET?
        BEQ     10$        ;NO, IF EQUAL
        DRSTS   D.RS22     ;RETURN STATUS
10$:    BIS     R0,(R1)    ;SET THE EVENT FLAG
        DRSTS   D.RS00     ;RETURN STATUS
20$:    DRSTS   D.RS97     ;ILLEGAL EVENT FLAG STATUS
        .END
```

Example 11-3   Processing Routine for New Directive

```
;+
;          $ FORM OF MACRO FOR $DRFLG
;

          .MACRO   SLEF$ TN,EFN
          .BYTE    161.,4
          R50$     TN
          .WORD    EFN
          .ENDM    SLEF$
;+
;          $C FORM OF MACRO FOR $DRFLG
;-
          .MACRO   SLEF$C TN,EFN,ERR,PST
          .PSECT   $DFB$$
$$$       =        .
          .BYTE    161.,4
          R50$     TN
          .WORD    EFN
          .PSECT   PST
          MOV      #$$$,-(SP)
          EMT      ^O<377>
          .IF NB ERR
          BCC      .+6
          JSR      PC,ERR
          .ENDC
          .ENDM    SLEF$C
;+
;          $S FORM OF MACRO FOR $DRFLG
;-
          .MACRO   SLEF$S TN,EFN,ERR
          MOV$     EFN
          RFA$     TN
          MOV      (PC)+,-(SP)
          .BYTE    161.,4
          DIR$     ,ERR
          .ENDM    SLEF$S
```

Example 11-4   Macros for New Directive

Adding a Directive to the System

- Directive can be included in the Executive or in EXCOM2

    In the Executive if

        Directive commons not chosen at SYSGEN

        Not enough room in EXCOM2

    Otherwise in EXCOM2

- To include in the Executive

    Build with the Executive during SYSGEN

- To include in EXCOM2

    Build with EXCOM2 during SYSGEN

- Macros included in system macro library RSXMAC.SML

MAC DRFLG=LB: [1,1] EXEMC/ML,SY: [11,10] RSXMC,DRFLG

LBR LB: [1,24] RSX11M/IN :DRFLG

LB: [1,24] RSX11M/LB:DRFLG

TKB @RSXBLD

```
┌──────────────┐
│  ASSEMBLE    │
│  DIRECTIVE   │
│  ROUTINE     │
└──────────────┘
      ┌──────────────┐
      │ ADD OBJECT   │
      │ MODULE TO    │
      │ LIBRARY      │
      └──────────────┘
            ┌──────────────────┐
            │ ADD LINE         │
            │ TO TASK BUILD    │
            │ COMMAND FILE     │
            └──────────────────┘
                  ┌──────────────┐
                  │  TASK BUILD  │
                  │              │
                  └──────────────┘
```
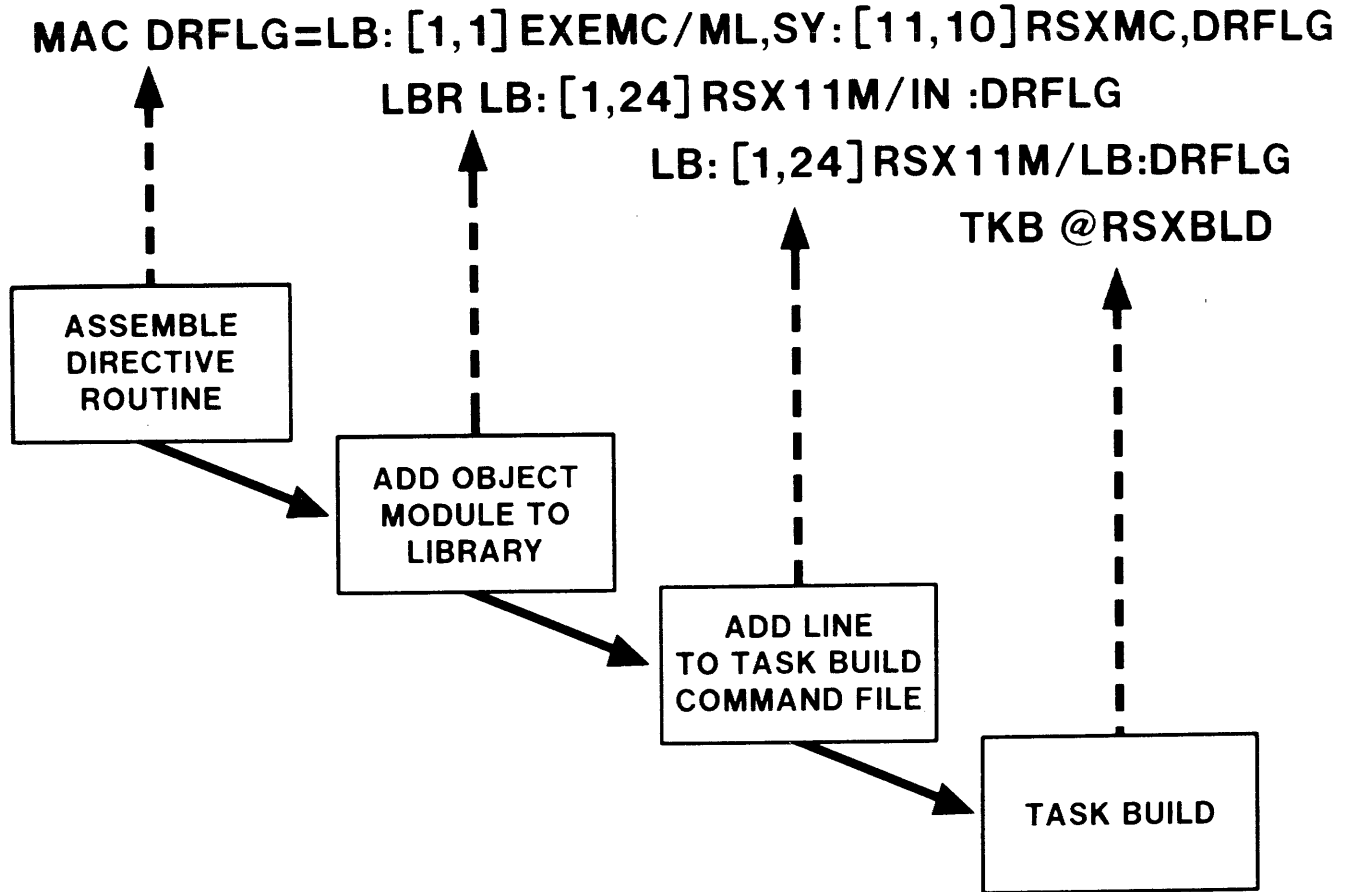
Figure 11-10 Assembling and Taskbuilding the Processing Routine

I/O PROCESSING

12

INTRODUCTION

Any modern operating system must provide a complete I/O system. Besides providing basic I/O device handling software, it must also provide a convenient file system.

Device drivers handle primitive I/O functions under RSX-11M. These drivers are tailored to individual devices and optimized to provide maximum performance. They are invoked by means of the QIO mechanism.

RSX-11M uses the FILES-11 file system. There are two levels of software support in the operating system for FILES-11. The more visible level includes the file access systems, FCS, and RMS. These systems provide all features needed in any application under RSX-11M. A more primitive level, in the form of ACPs (F11ACP), forms the interface between FCS/RMS and device drivers.

OBJECTIVES

1. Diagram the flow in the I/O system.

2. List the types of ACPs and their differences.

3. List the steps in processing an I/O request within DRQIO.

RESOURCES

1. RSX-11M System Lists and Data Structures

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

3. RSX-11M/M-PLUS I/O Operations References Manual

OVERVIEW

Components of the I/O System

- The File System

    Used to structure data for convenient use in applications

    Provides software for easy manipulation of data

    Consists of:

    FILES-11 structures

    File Control Services (FCS)

    Record Management Services (RMS)

- QIO directive

    Used to request I/O service from the Executive

    Can be used directly in user task

    All file-system I/O requests converted into QIOs

- Device Drivers

    Used to control physical devices and perform actual I/O

    One required for each device type

    Carry I/O requests to transfer data

- Ancillary Control Processor (ACP)

    Performs functions subordinate to those of the driver

        Provides a file system for user tasks

        Handles protocols needed by a communications network

    Converts I/O requests into one or more request to a device driver

    Manipulates I/O parts of the system data base

    Three ACPs provided with RSX-11M

        F11ACP: Files-11

        MTAACP: Magtape I/O ACP

        NETACP: DECNET ACP

USER STATE | SYSTEM STATE

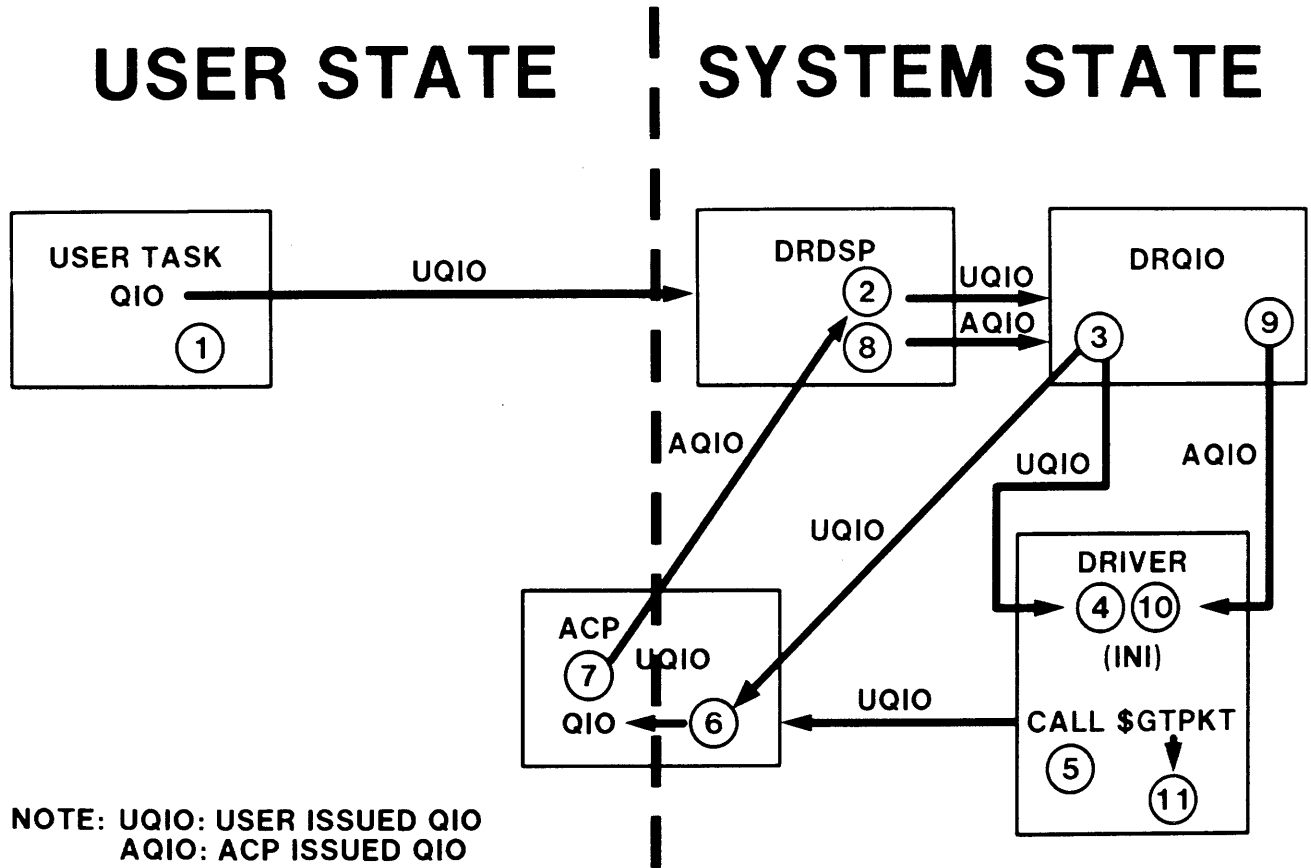NOTE: UQIO: USER ISSUED QIO
AQIO: ACP ISSUED QIO

Figure 12-1 The I/O Process

Initiating I/O From a User Task

- Use QIO directives

    Harder to use than other methods

    Can be the most efficient method of I/O

    For file I/O, use virtual block transfers

    For non-file I/O, use logical block transfers

- Use high-level language I/O commands

    Easier to use than QIOs, but less efficient

    Language processor may convert to QIOs or convert to FCS/RMS commands

- Use FCS/RMS to perform file system I/O

    Allows data to be accessed in a form natural to the application

    File processor converts to QIOs

Processing I/O Request in DRQIO

- All I/O performed as a result of QIO directives

- QIO directive parameters are checked; directive status is returned

- Parameters used to create I/O packet

- I/O packet is queued to a device driver or to an ACP

- If ACP function, a number of checks are performed before queuing I/O packet

QIO PROCESSING

Logical Units

- Logical Unit Numbers (LUNs) identify the device to which I/O is to be performed

- LUN determines a unique entry in the task's Logical Unit Table (LUT) in the task header

- Each entry in the LUT is two words long

- When a task is installed, the first word of each LUT entry is initialized with the UCB address of the device

- When a file is opened on a volume, the second word of the corresponding LUN points to a FILES-11 window block
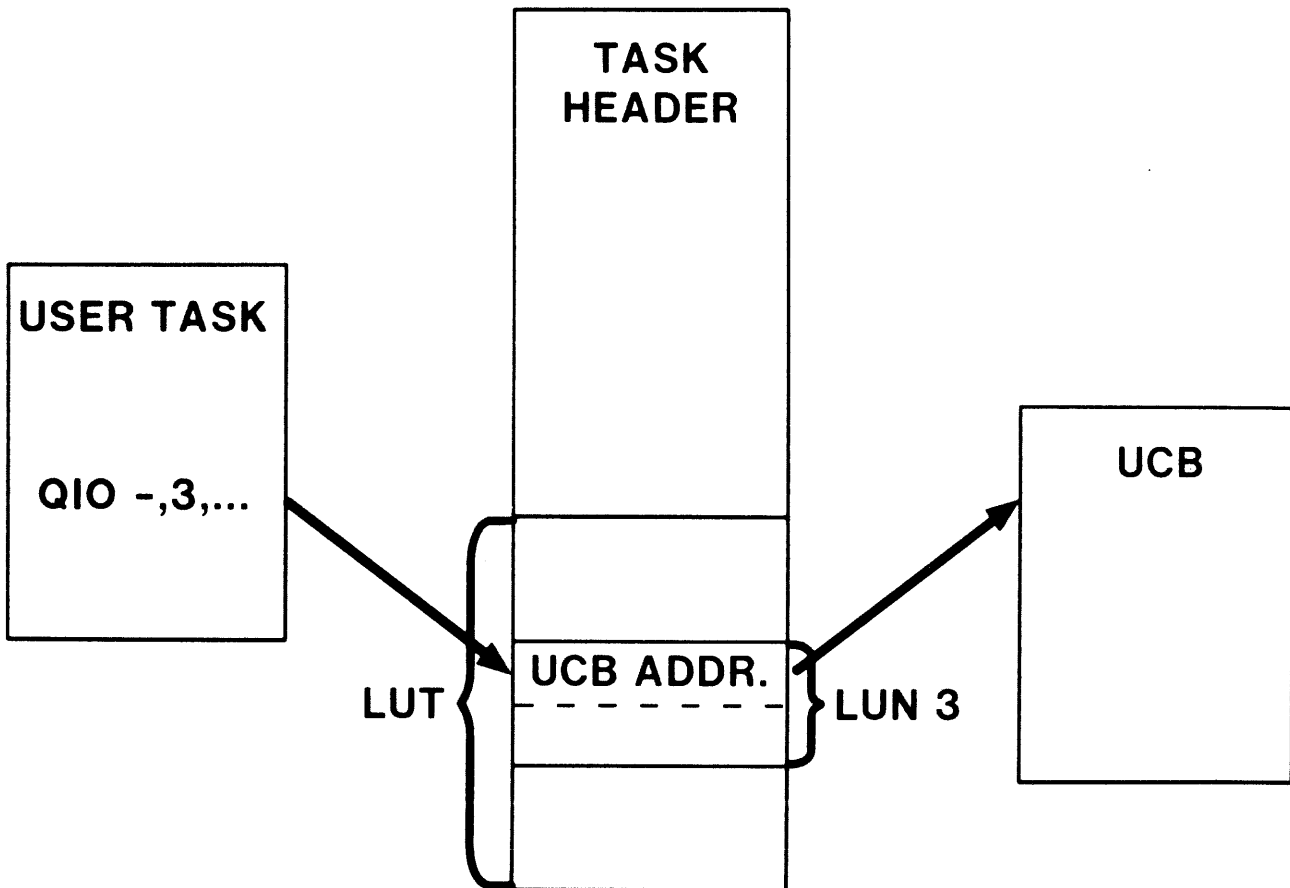
Figure 12-2   Logical Units

Processing Within DRQIO

- Locates the UCB for LUN

    Calls $MPLUN in module EXESB

- Checks to see if driver resident

    Location D.DSP in the DCB

- Clears specified event flag

- Clears I/O status block if specified

- QIO directive is now successful

    DSW shows success

    Any errors are reported as I/O errors in the IOSB

- Allocates space for I/O packet in POOL

    Calls $ALPKT in module CORAL

- Increments I/O request count in the TCB

- Copy data from DPB to I/O packet

- If function is IO.KIL, go the routine $IOKIL in module IOSUB

- Check if task has privileges to perform I/O

- Check function code against function masks in DCB

- If not legal function, return I/O error

- If control function, queue I/O packet to driver

- If 'no-op' function, return successful status

- If ACP function to mounted device, perform ACP function checks

The ACP Function Checks

- Implemented by a collection of primitive routines

- Each ACP requires a different group of these routines

- A two-level dispatch table, the Polish Interpreter Database, is used

- First level table, the Function Code Dispatch Vector, resides at FCDSP in DRQIO

    Each entry corresponds to a unique function code

    Each entry is a vector which points to a second level dispatch table

- Second level dispatch table contains primitive routines to be called

Table 12-1   Function Code Dispatch Vector

Function Code	Dispatch Table	Function
10	FCIFC	Illegal function
11	FCPKT	Find file name in directory
12	UNLCK	Unlock Block
13	FCPKT	Remove file name from directory
14	FCPKT	Enter file name in directory
15	FCACC	Access file for READ
16	FCACC	Access file for READ and WRITE
17	FCACC	Access file for READ, WRITE, and EXTEND
20	FCDAC	Deaccess file
21	FCRVB	Read Virtual Block
22	FCWVB	Write Virtual Block
23	FCEXT	Extend file
24	FCCRE	Create file
25	FCDEL	Mark file for DELETE/TRUNCATE file
26	FCPKT	Read file attributes
27	FCPKT	Write file attributes
30	FCPKT	User magtape control function
31	FCWVB	Transmit process message
32	FCRVB	Receive process message
33	FCCON	Connect to process
34	FCDIS	Disconnect from process
35	FCNCT	Network control process
36	FCIFC	Illegal function
37	FCIFC	Illegal function

Table 12-2   Polish Routines

Table	Routine	Function
FCIFC	IEIFC	Set illegal function status
FCACC	CKDMO	Check if volume marked for dismount
	CKALN	Check if file already accessed on LUN
	CKMOU	Check volume mounted this user
	CKACP	Set flag to queue to ACP
	BDPKT	Build an I/O packet
	CKRLK	Synchronize access and exit
FCDAC	CKNLN	Check if file accessed on LUN
	CKQMT	Queue to MTAACP if ANSI
	BDPKT	Build an I/O packet
	CKRLK	Synchronize and exit
FCRVB	CKNLN	Check if file accessed on LUN
	CKRAC	Check read access and exit
FCWVB	CKNLN	Check if file accessed on LUN
	CKWAC	Check write access and exit
FCCRE	CKDMO	Check if volume marked for dismount
	CKALN	Check if file accessed on LUN
	CKPKT	Join common ACP and check mount
FCDEL	CKTRN	Check if really truncate
	CKACP	Set flag to queue to ACP
	CKMOU	Check volume mounted by this user
	BDPKT	Build an I/O packet
	CKXIT	Exit
FCPKT	CKACP	Set flag to queue to ACP
	CKMOU	Check volume mounted by this user
	BDPKT	Build an I/O packet
	CKXIT	Exit
FCEXT	CKPND	Increment window pending count for extend
UNLCK	CKNLN	Check if file accessed on LUN
	UNLXT	Set registers and exit
FCCON	CKDMO	Check if volume marked for dismount
	CKALN	Check if process already connected on LUN
	CKCON	Address check connect buffer
	CKRLK	Interlock LUN usage and exit

Table 12-2   Polish Routines (Cont)

Table	Routine	Function
FCDIS	CKNLN	Check if process connected on LUN
	CKDIS	Check buffer and copy parameters
	CKRLK	Interlock LUN usage and exit
FCNCT	CKDMO	Check if volume marked for dismount
	CKCON	Address check buffer
	CKXIT	Exit

Queuing I/O Requests

- I/O request packets are queued by routine $DRQRQ in module DRSUB

- Processing in $DRQRQ

  Bit UC.QUE in U.CTL in the UCB is checked

  If set, packet queued to driver

  Contents of KERNEL APR 5 is saved on stack

  If driver loadable, base address of driver loaded into KERNEL APR 5

  The driver is called to initiate I/O

  When driver exits

  Control returns to $DRQRQ

  Original contents of KERNEL APR 5 are restored

ANCILLARY CONTROL PROCESSORS

Types of ACPs

● Standard ACPs

    Known to the Executive

    Executive includes special code for support

    Follow rules used by DEC in writing F11ACP and MTAACP

● User ACPs

    Not known to the Executive

    Known only to the driver which uses their services

    UC.QUE set in U.CTL to prevent ACP function checks in DRQIO

    Driver does preprocessing and queues requests to ACPs

    No change to the Executive code is necessary

● Foreign ACPs

    Between standard and user ACPs

    Known to the Executive

    No special services provided to support ACP
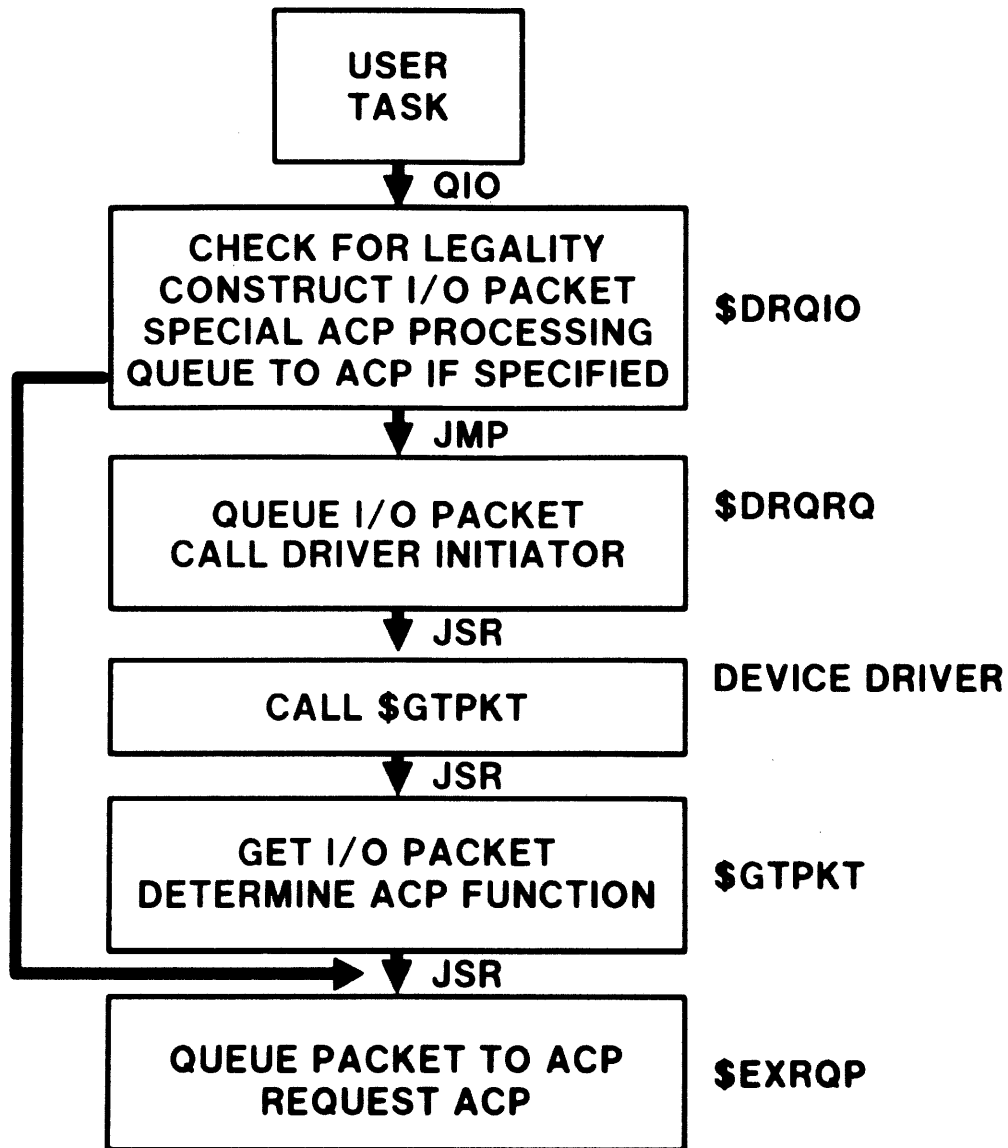
    Identified by foreign bit US.FOR in the UCB

Figure 12-3   Standard ACP

Figure 12-4 User ACP
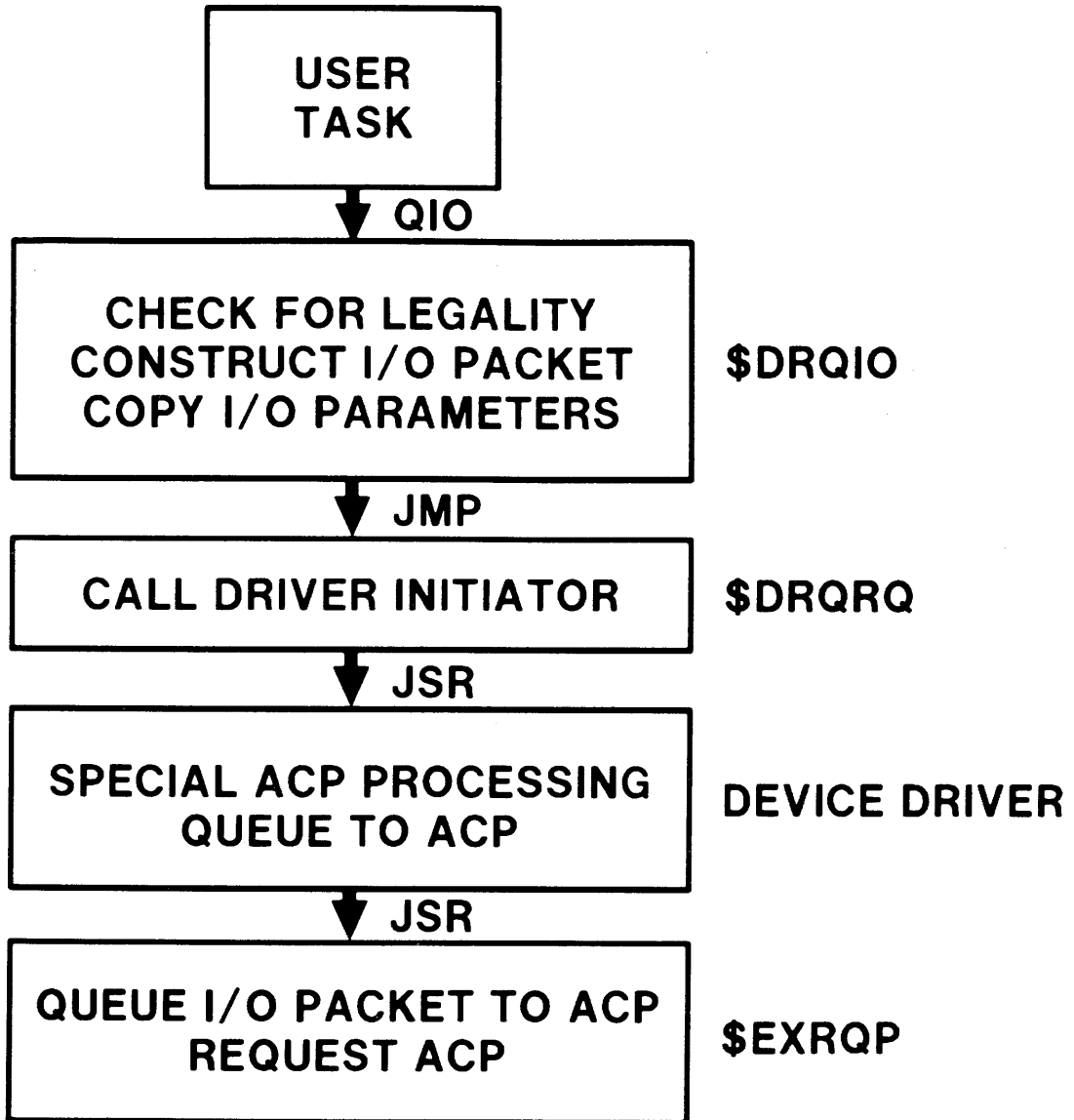
```
                    ┌─────────────────┐
                    │      USER       │
                    │      TASK       │
                    └─────────────────┘
                             │ QIO
                             ▼
          ┌──────────────────────────────────┐
          │    CHECK FOR LEGALITY             │
          │    CONSTRUCT I/O PACKET           │   $DRQIO
          │    COPY I/O PARAMETERS            │
          │    QUEUE TO ACP IF UC.QUE=0       │
          └──────────────────────────────────┘
                             │ JMP
                             ▼
          ┌──────────────────────────────────┐
          │    CALL DRIVER INITIATOR          │   $DRQRQ
          └──────────────────────────────────┘
                             │ JSR
                             ▼
          ┌──────────────────────────────────┐
          │    SPECIAL ACP PROCESSING        │
          │    QUEUE I/O PACKET              │   DEVICE DRIVER
          │    CALL $GTPKT                    │
          └──────────────────────────────────┘
                             │ JSR
                             ▼
          ┌──────────────────────────────────┐
          │    GET I/O PACKET                │
          │    DETERMINE ACP FUNCTION         │   $GTPKT
          └──────────────────────────────────┘
                             │ JSR
                             ▼
          ┌──────────────────────────────────┐
          │    QUEUE PACKET TO ACP           │
          │    REQUEST ACP                    │   $EXRQP
          └──────────────────────────────────┘
```
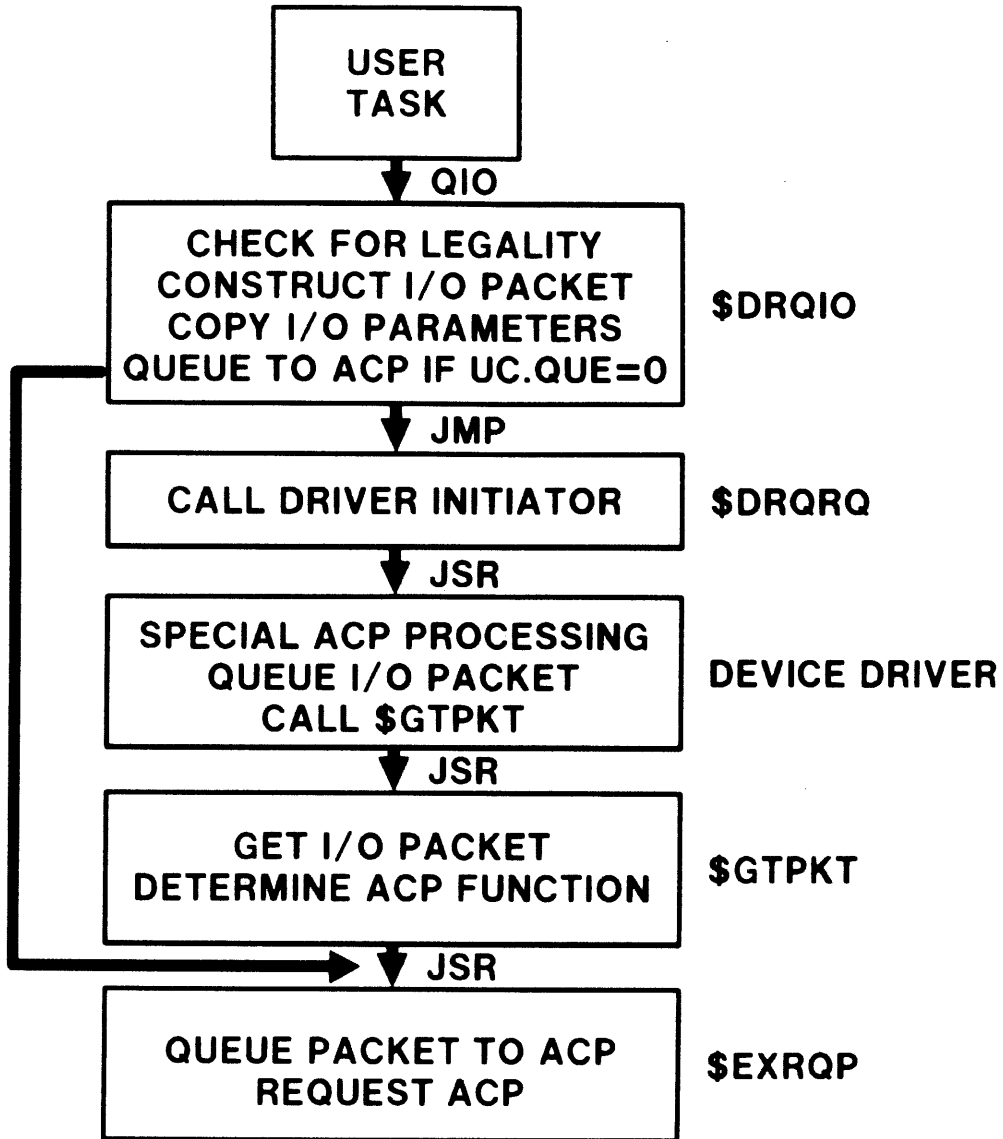
Figure 12-5 Foreign ACP

The FILES-11 (F11ACP)

- Implements the FILES-11 structure for disks

- Provides needed file primitives (open, close, etc.)

- Converts file primitives to driver primitives

    Open file requires several driver operations:

        Locating the file which may require several reads of directory records and headers

        Creating the FCB from the file header

        Creating the window block from the retrieval pointers

Versions available at SYSGEN

- MINIMUM Version

    Not recomended for general use

    Runs in 2K words of memory

    Heavily overlaid

    No internal buffers for storing data

    Mapped and unmapped systems

- SMALL Version

    Provides limited performance optimization

    Requires 10 to 100 times more I/O for directory lookups and file opens than larger ACPs

    Heavily overlaid

    Runs in 2.5K words of memory

    Mapped and unmapped systems

    Default version for unmapped systems

● MIDDLE Version

> Usually provides the best performance without using large amount of memory
>
> Provides buffers for storing directories and the index file bitmap
>
> Runs in 5K words of memory
>
> Mapped and unmapped systems
>
> Default version for mapped systems

● LARGE Version

> Provides the maximum performance for F11ACP
>
> Not overlaid
>
> Runs in 10K words of memory
>
> Provides larger buffers for storing directories
>
> Available only on mapped systems

DEVICE DRIVER PROCESSING

13

INTRODUCTION

Most tasks receive raw data from external sources and provide results to the user. It is usually necessary to keep large amounts of data available to the processor for instant use. To provide these services, an I/O system is needed. The basic component of this system is the software which handles the activities of the physical devices. Under RSX-11M, this software is called a device driver.

RSX-11M provides drivers for each supported peripheral device. The interface between a driver and the I/O system is simple and standard. The user can write drivers for non-standard devices by adhering to the requirements of this interface.

OBJECTIVES

1. Diagram the flow of control during device driver processing.

2. Determine the status of a device by examining its data structures.

3. Diagram the process of performing I/O with a DMA UNIBUS device on the PDP-11/44 or 70.

RESOURCES

1. RSX-11M System Lists and Data Structures

2. RSX-11M Guide to Writing a Device Driver

OVERVIEW

Methods of Performing I/O

- Device common

    Partition in I/O Page

    Built and used as any other common area

    Can only perform 'device-poll' device driving

- Direct access to I/O page

    Privileged task has access to I/O page

    Can only perform 'device-poll' device driving

- CINT$ Directive

    Task contains essential routines to drive device

    Very little knowledge of system interface required

    Easy to debug

    Interrupt driven

- Device Driver

    Integral part of the Executive

    Is written to use QIO mechanism

    Usually interrupt driven but can do 'device-poll'

Design Philosophy

- One device driver per device type

    Handles multiple controllers

- Common I/O processing done in Executive routines

    Queuing of I/O request in module DRQIO

    I/O completion handled by routine $IODON in module IOSUB

- Keep interrupt lockout to a minimum

    Run at priority 7, no more than 100 microseconds

    Run at priority of interrupting device, no more than 500 microseconds

- Use a FORK process to access the system data base

General Flow

● User task issues QIO directive

    Executive entered as result of EMT instruction

● Executive processes QIO

    Checks made on data in DPB

    Mapping adjusted to map to service routine and DPB

● Module DRQIO processes QIO

    Checks made on I/O request

    I/O packet created and queued to driver

● Driver called at initiate I/O entry point

    I/O packet retrieved

    Data in packet processed

    I/O device started

● Control returns to user task if asynchronous I/O, Executive otherwise

● When interrupt occurs, driver entered at interrupt entry point

DRIVER DATA BASE

- Location

    In module SYSTB for DEC-supplied drivers

    If user-written driver with

        Resident data base, in Executive
        Loadable data base, in DSR

Device Control Block (DCB)

- One per device type

- Describes static characteristics of device

- Linked to device list

    Headed by $DEVHD in Executive module SYSCM

    List linked by D.LNK in DCB

- Contains

        D.DSP   - Address of Driver Dispatch Table

        D.MSK   - Function masks for filtering QIO requests

        D.UCB   - Link to first UCB

        D.UCBL   - Length of UCB for device type

        D.NAM   - Generic device name in ASCII

        D.UNIT   - Highest and lowest unit number

- Used by QIO processing, not by driver

Unit Control Block (UCB)

- One per physical unit

- Contains mostly static data

- Most other data structures can be accessed from UCB

- UCB for unit 0 linked from D.UCB in the DCB for device type

- Other UCBs for device type contiguous with UCB for unit 0

- Contains

    U.DCB - Pointer back to DCB for device type

    U.RED - Pointer to UCB of redirect device

    U.UNIT - Physical unit number

    U.SCB - Pointer to SCB of controller for this unit

- Used by both driver and Executive

Table 13-1   Important Entries in the UCB

Entry	Offset	Length	Contents
U.DCB	0	WORD	Address of DCB for device type
U.SCB	20	WORD	Address of SCB for unit
U.UNIT	6	BYTE	Physical unit number
U.RED	2	WORD	Address of UCB for redirect
U.STS	5	BYTE	Unit status bits
U.ST2	7	BYTE	More unit status bits
U.CTL	4	BYTE	Unit control flags
U.CW1	10	WORD	1st device characteristics word
U.CW2	12	WORD	2nd device characteristics word
U.CW3	14	WORD	3rd device characteristics word
U.CW4	16	WORD	4th device characteristics word
U.CLI	-6	WORD	Default CLI
U.LUIC	-4	WORD	Login UIC
U.OWN	-2	WORD	Owning terminal

Notes: 1.   U.CLI and U.LUIC exist only for terminals
         1.   U.LUIC exists if system is multi-user

	.DCØ		0 U.DCB
	.LPØ		2 U.RED
U.STS	0	0	4 U.CTL
U.ST2	0	0	6 U.UNIT
	DV.REC ! DV. CCL		10 U.CW1
	140100		12 U.CW2
	0		14 U.CW3
	132.		16 U.CW4
	$LPØ		20 U.SCB
	0		22 U.ATT
	0		24 U.BUF
	0		26
	0		30 U.CNT

Figure 13-1   Unit Control Block for LP:

Offset	Label			Byte	Name
		0		-10	U.IOC
U.ERHL	8.		5	-6	U.ERSL
U.ERHC	0		0	-4	U.ERSC
		0		-2	U.OWN
		.DCO		0	U.DCB
		.DKO		2	U.RED
U.STS	UC.ALG!UC.NPR!UC.PWF		US.MNT	4	U.CTL
U.ST2	0		0	6	U.UNIT
	DV.DIR!DV.MNT!DV.F11!DV.MSD!DV.UMD			10	U.CW1
		0		12	U.CW2
		4800.		14	U.CW3
		512.		16	U.CW4
		$DKO		20	U.SCB
		0		22	U.ATT
		0		24	U.BUF
		0		26	
		0		30	U.CNT
		0		32	U.ACP
		0		34	U.VCB

Figure 13-2 Unit Control Block for DK:

Status Control Block (SCB)

- One SCB per controller

- Line multiplexers (DH-11, DJ-11, etc.) have one controller per line

- Most information is dynamic

- Linked from U.SCB in the UCB

- One SCB is linked from several UCBs if controller is common to several units

- Contains

    S.LHD - Device I/O queue listhead

    S.PKT - Address of current I/O packet

    S.PRI - Device priority

    S.CTM - Current time out count

    S.ITM - Initial time out count

    S.CSR - Address of Control Status Register

    S.FRK - The FORK block

- Used by driver and Executive

S.ROFF -7	REGS. TO COPY ON ERROR	OFFSET TO 1ST REG.	**-6** **S.RCNT**
	SAVED I/O ACTIVE BIT MAP		**-4** **S.BMSV**
	DEVICE I/O ACTIVE BIT MASK		**-2** **S.BMSK**
	CONTROLLER I/O		**0** **S.LHD**
	QUEUE LISTHEAD		**2**
S.VCT 5	VECTOR ADDR/4	PRIORITY	**4** **S.PRI**
S.ITM 7	INITIAL TIMEOUT COUNT	CURRENT TIMEOUT COUNT	**6** **S.CTM**
S.STS 11	CONTROLLER STATUS	CONTROLLER INDEX	**10** **S.CON**
	ADDRESS OF CSR		**12** **S.CSR**
	ADDRESS OF CURRENT I/O PACKET		**14** **S.PKT**
	FORK BLOCK LINK WORD		**16** **S.FRK**
	FORK - PC		**20**
	FORK - R5		**22**
	FORK - R4		**24**
	FORK-DRIVER RELOCATION BASE		**26**
	LINK WORD		**30** **S.MPR/M.LNK**
	ADDRESS OF FIRST ASSIGNED UMR		**32** **M.UMRA**
UMR ASSIGNMENT BLOCK	NUMBER OF UMR'S ASSIGNED ·4		**34** **M.UMRN**
	LOW 16 BITS MAPPED BY 1ST UMR		**36** **M.UMVL**
M.UFVF	HIGH 6 BITS OF BUFFER	HIGH 2 BITS MAPPED	**40** **M.UMVH**
	LOW 16 BITS OF PHYSICAL BUFFER ADDRESS		**42** **M.VFVL**

Figure 13-3   Status Control Block

I/O Packet

- Used to communicate between Executive and device driver

- Created by Executive in DSR

- One for each I/O request queued to driver

- Each SCB has an I/O queue consisting of pending I/O packets

    Linked from listhead S.LHD in the SCB

    Linked in order of decreasing task priority

    Packet is removed from queue when service begins

- Current I/O packet is linked from S.PKT in the SCB

- Contains

    I.PRI   - Priority of request

    I.EFN   - Event flag to use for synchronization

    I.TCB   - Address of TCB of requesting task

    I.FCN   - I/O function code

    I.IOSB   - I/O status block address

    I.AST   - Address of AST service routine

    I.PRM   - I/O parameter list

I/O QUEUE LINK WORD	0	I.LNK
EVENT FLAG NUMBER / REQUEST PRIORITY	2	I.PRI
TCB ADDRESS OF REQUESTER	4	I.TCB
POINTER TO SECOND LUT WORD	6	I.LN2
POINTER TO UNIT CONTROL BLOCK	10	I.UCB
MAIN FUNCTION CODE / MODIFIER	12	I.FCN
VIRTUAL ADDRESS OF I/O STATUS BLOCK	14	I.IOSB
RELOCATION BIAS (32₁₀ WORD BLOCK ADDRESS)	16	
KERNEL VIRTUAL ADDRESS (140000 + DIB)	20	
VIRTUAL ADDRESS OF AST ROUTINE	22	I.AST
RESERVED FOR MAPPING PARAMETER 1	24	I.PRM

I.EFN 3 appears to the left of the EVENT FLAG NUMBER / REQUEST PRIORITY row.

PARAMETERS	26
1-6	30
	32
	34
	36
	40

Figure 13-4   I/O Packet
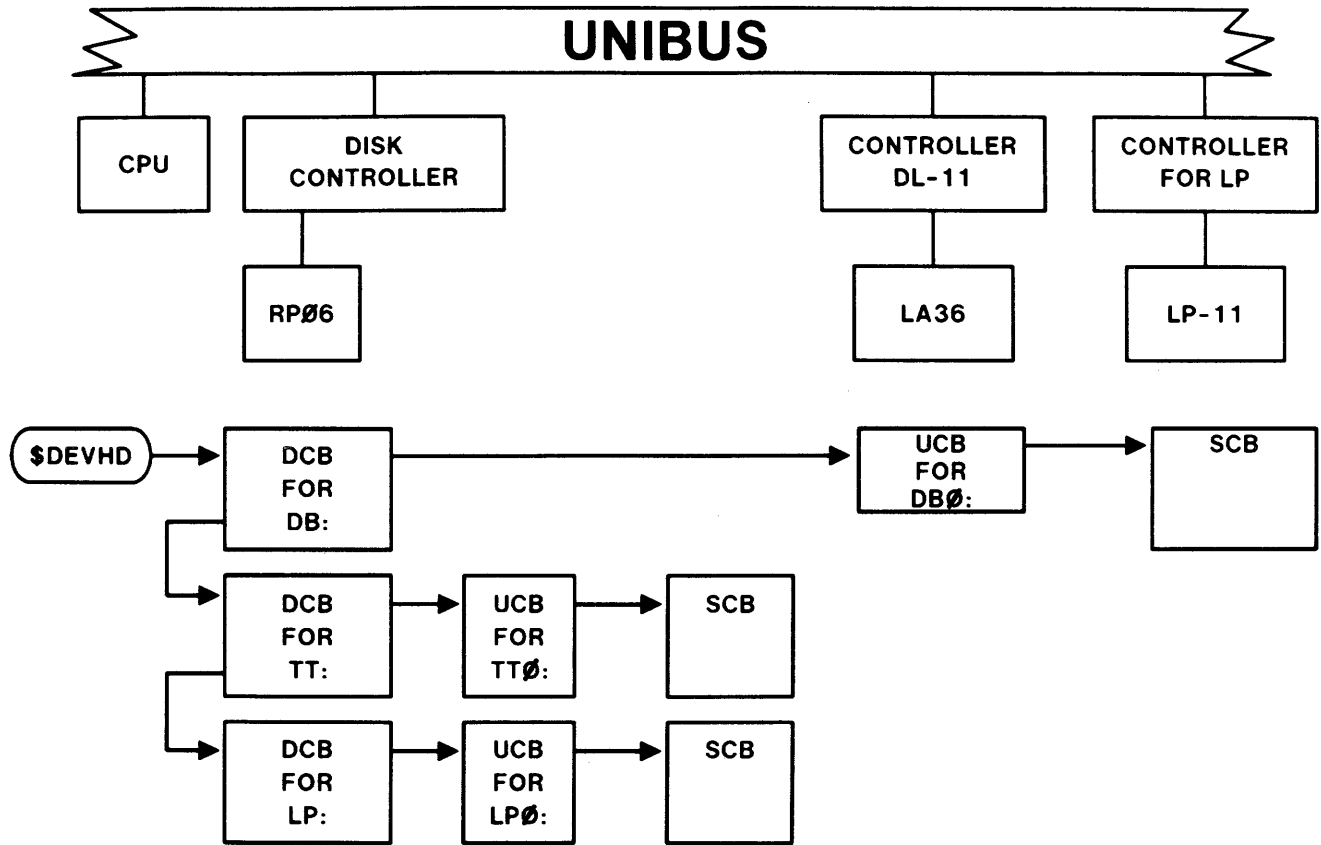
Figure 13-5   A Simple Configuration
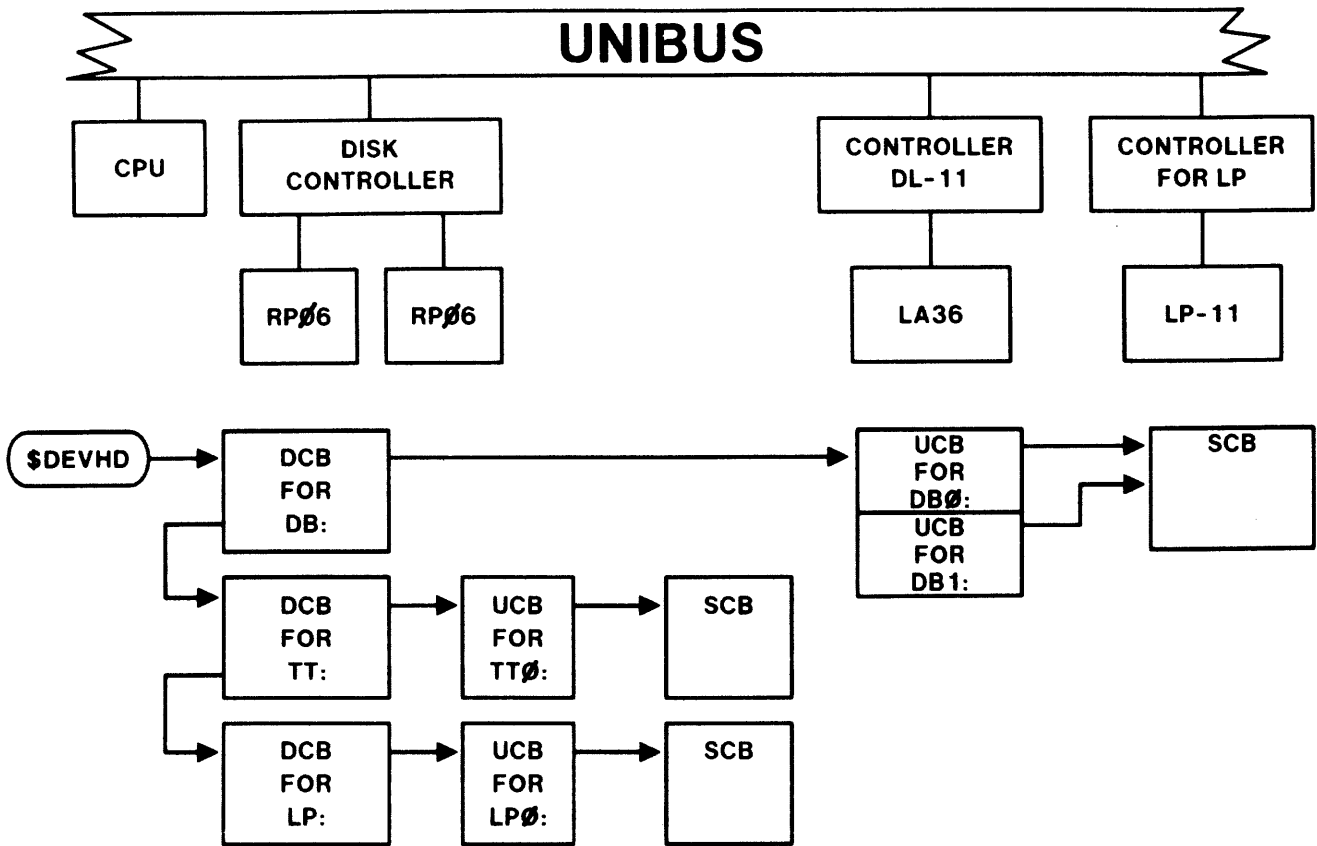
Figure 13-6 A Configuration with Multiple Units
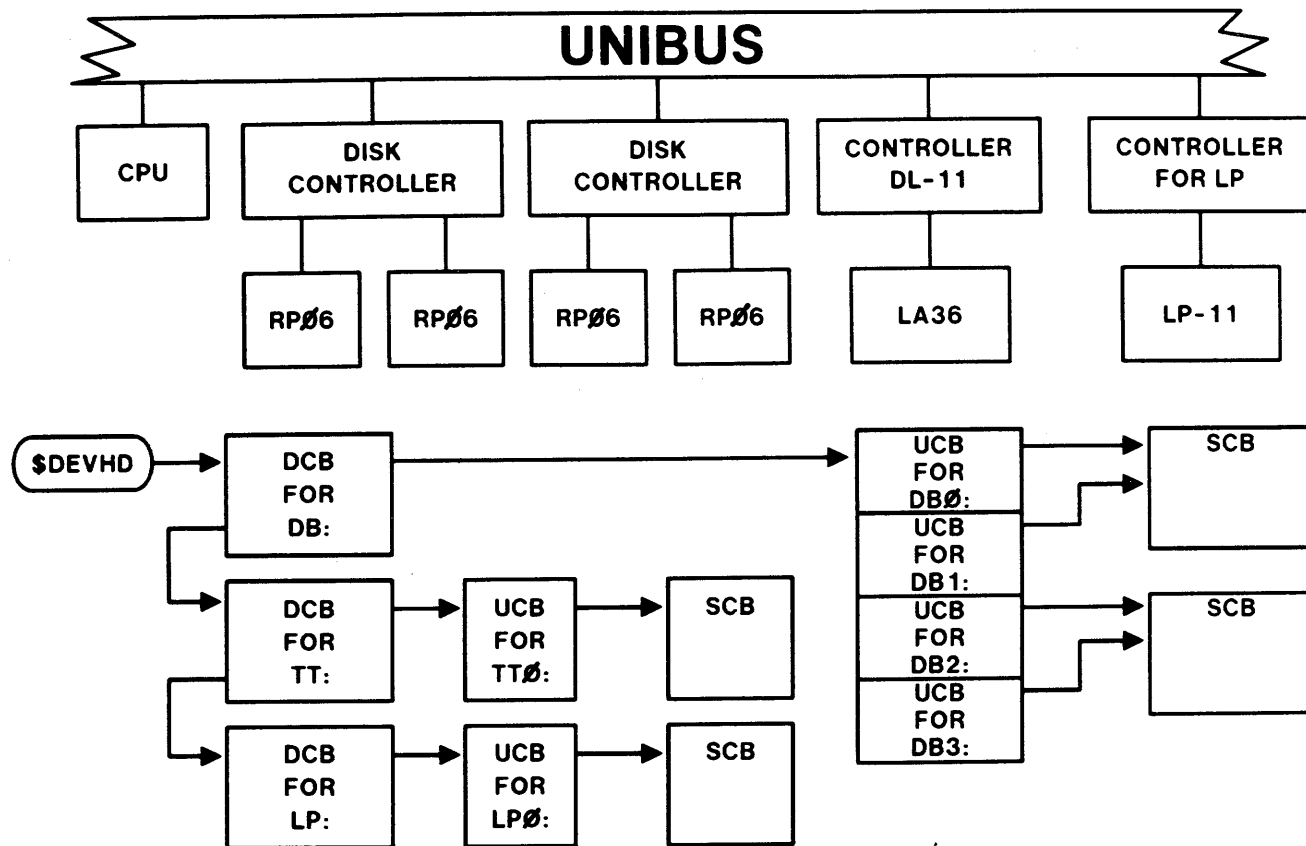
Figure 13-7   A Configuration with Multiple Controllers

DRIVER STRUCTURE

Driver Entry Points

- Initiate I/O Entry Point

    Used to initiate an I/O operation
    Called from routine $DRQIO
    Executes at processor priority 0
    Name of entry point: XXINI

- Kill I/O Entry Point

    Cancels in-progress requests for non block-addressable devices

    Called from routines $DRQIO and $DREIF

    Executes at device priority

    Name of entry point: XXCAN

- Time-out Entry Point

    Used when driver does not interrupt within specified time interval after a request

    Called from module TDSCH

    Executes at device priority

    Action: Retry I/O operation

    After eight retries, I/O operation is aborted

    Name of entry point: XXTMO

- Powerfail Entry Point

    Called by module INITL and by SAVE to initialize device
    Reinitializes device after powerfail
    Called from routine POWER after power is restored
    Executes at processor priority 0
    Name of entry point: XXPWF

● Interrupt Entry Point

    Services interrupts from device
    Address stored in interrupt vector
    Entered at processor priority 7
    Entry Point Names Critical

        XXINT - Single interrupt driver
        XXINP - Input, two interrupt drivers
        XXOUT - Output, two interrupt drivers

Driver Dispatch Table

● Table containing addresses of service routines for all entry
  points except the interrupt entry

● Table is in the device driver at location $XXTBL

```
$XXTBL::
            .WORD    XXINI    ;Initiate I/O
            .WORD    XXCAN    ;Cancel I/O
            .WORD    XXTMO    ;Device Timeout
            .WORD    XXPWF    ;Powerfail
```

● Pointed to by D.DSP in the DCB

● Entries are local symbols whose order is critical

Figure 13-8   Driver Flow
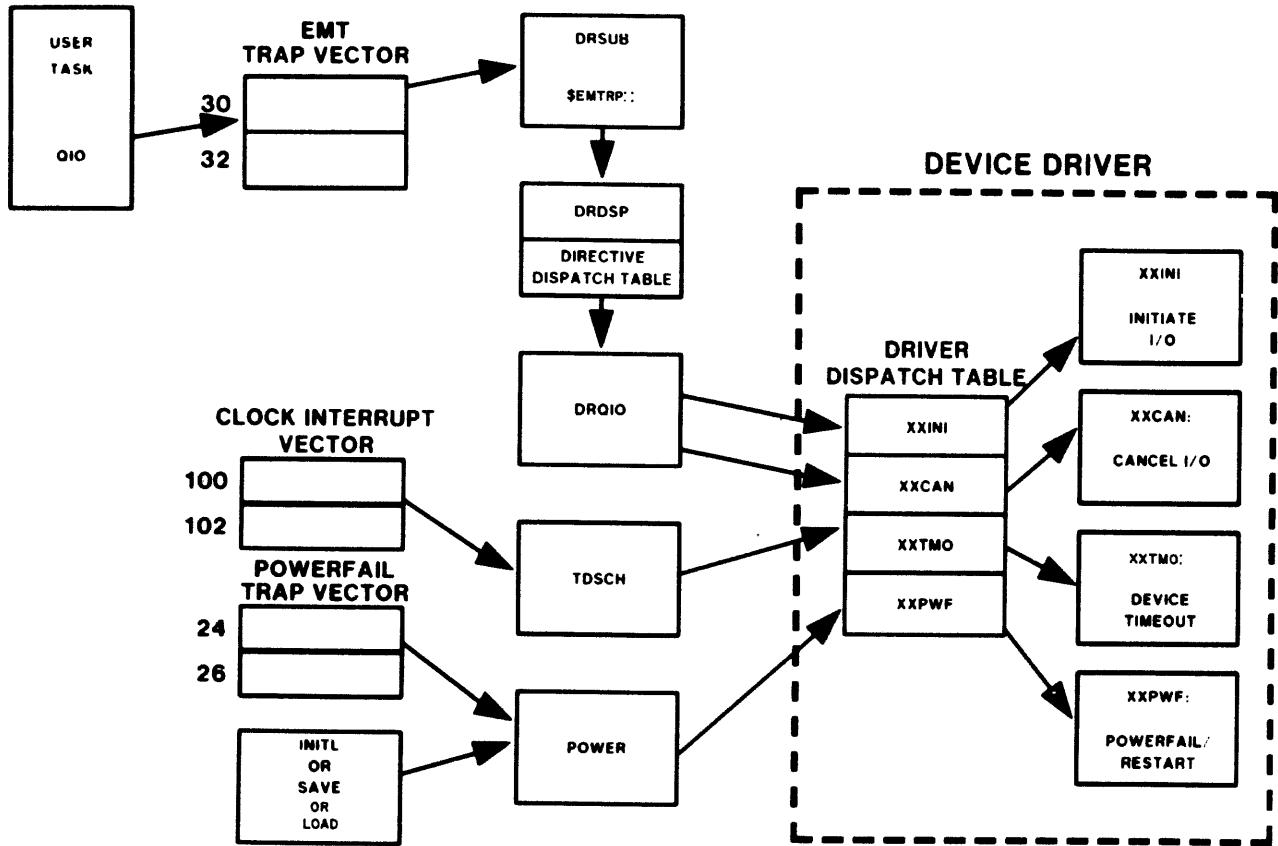
```
          .TITLE   XXDRV
$XXTBL::
          .WORD    XXINI
          .WORD    XXCAN
          .WORD    XXTMO
          .WORD    XXPWF

XXINI:    CALL     $GTPKT
          BCC      .+4
          RETURN
          MOV      #IS.SUC&377,R0
          CLR      R1
          CALL     $IODON
XXCAN:
XXTMO:
XXPWF:
XXINT::   RETURN
          .END
```

Example 13-1 A Skeleton Driver

```
                .TITLE   XXTAB
                .MCALL   DCBDF$,UCBDF$
                DCBDF$
                UCBDF$
        $XXDAT::
        XXDCB:   .WORD   0
                 .WORD   .XX0
                 .ASCII  /XX/
                 .BYTE   0,0
                 .WORD   XXND-XXST
                 .WORD   0
                 .WORD   177777
                 .WORD   30
                 .WORD   0
                 .WORD   177740
                 .WORD   377
                 .WORD   0
                 .WORD   0
                 .WORD   377
                 .WORD   0


        XXST=.
        .XX0::   .WORD   XXDCB
                 .WORD   .-2
                 .BYTE   0,US.MNT,0,0
                 .WORD   DV.MNT!DV.F11
                 .WORD   0,0
                 .WORD   80.
                 .WORD   $XX0
                 .WORD   0,0,0,0
                 .WORD   0                       ;U.ACP
                 .WORD   0                       ;U.VCB
        XXND=.

        $XX0::   .WORD   0
                 .WORD   .-2
                 .WORD   0,0,0,0,0,0,0,0,0

        $XXEND::
                 .END
```

Example 13-2   A Skeleton Driver Data Base

DRIVER PROCESSING

Initiate I/O

- Call $GTPKT to get an I/O packet

    Carry set on return from $GTPKT

        No packet

        ACP function

        Controller busy

- Registers on return from $GTPKT

    R1 - Address of I/O packet

    R2 - Physical unit number

    R3 - Controller index (into CNTBL)

    R4 - Address of SCB

    R5 - Address of UCB

- UCB address saved in CNTBL

- Perform initiation of I/O including

    Allocating UMRs if a DMA device on 11/44 or 11/70

    Setting up device timeout

    If error logging, call $BMSET

    Checking function code

    Performing operations on CSR(s)

    Mapping UMR(s) if required

    Enabling interrupts

Kill I/O

- Registers when entry point called

      R5 – Address of UCB

      R4 – Address of SCB

      R3 – Controller index

      R1 – TCB of current task

      R0 – Active I/O packet

- Processor is at device priority when called

- Cancel current I/O operation by

      Zeroing byte-count or flagging abort

      Canceling timeout by zeroing S.CTM in the SCB

- For DMA devices, current operation may not be canceled

Device Timeout

- Registers when entry point called

    R5 - Address of UCB

    R4 - Address of SCB

    R3 - Controller index

    R2 - Address of CSR

    R0 - I/O status code (IE.DNR)

- Processor at device priority when called

- Generate error message by calling $DVMSG with T.NDNR in R0

- If error logging, call $DTOER

- Reset timeout count if required

- Retry I/O operation required number of times

Powerfail

- Registers when entry point is called

    R5 - Address of UCB

    R4 - Address of SCB

    R3 - Controller index

- Processor at processor priority 0 when called

- Uses timeout mechanism for device recovery

    Special timeout count is used for many devices

    Performs other initialization tasks (e.g., the KMC-11 for the lineprinter)

Interrupt Processing

- Registers R4 and R5 saved on entry, others cannot be used except in FORK process code

- Processor priority

    At call: 7

    After return from $INTSV: Device priority

    During FORK processing: 0
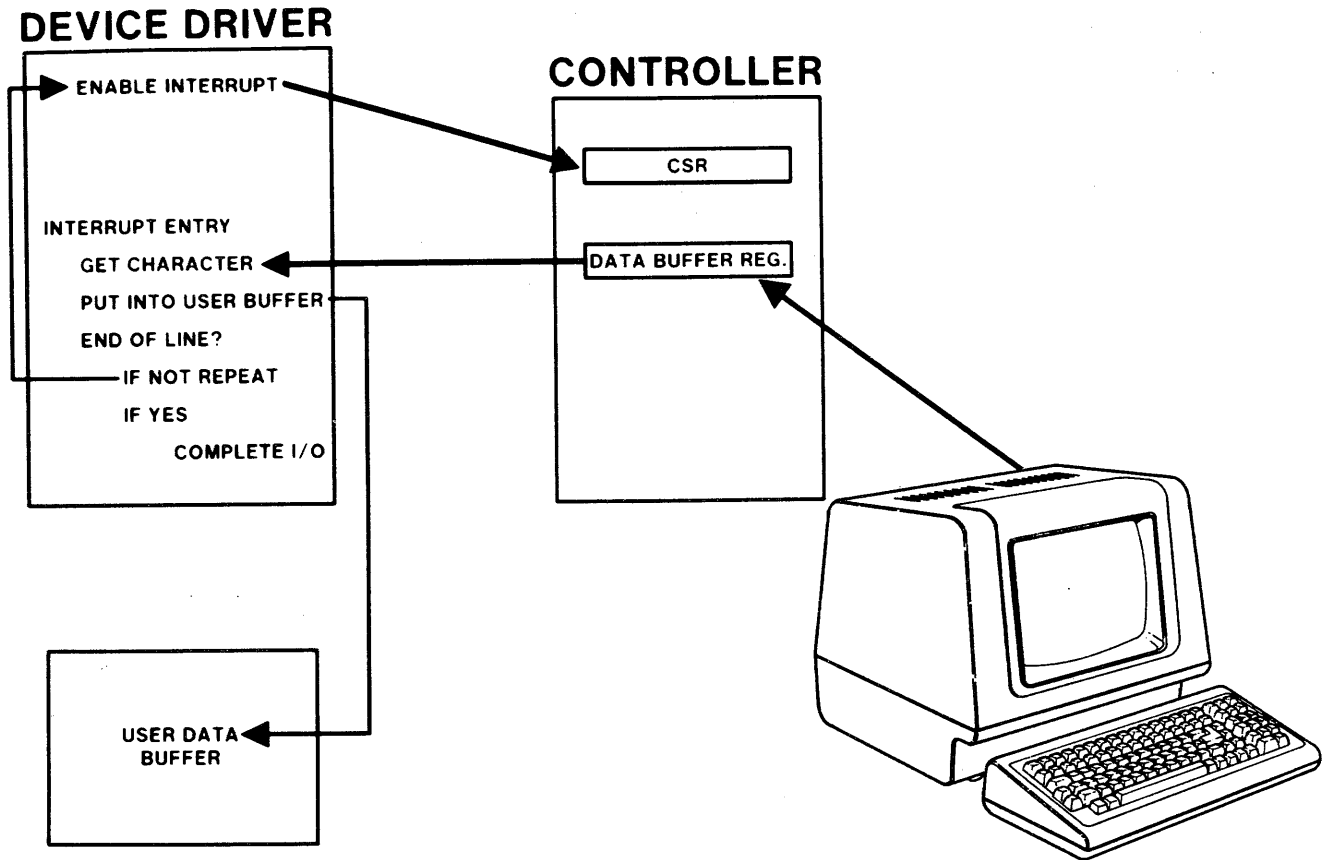
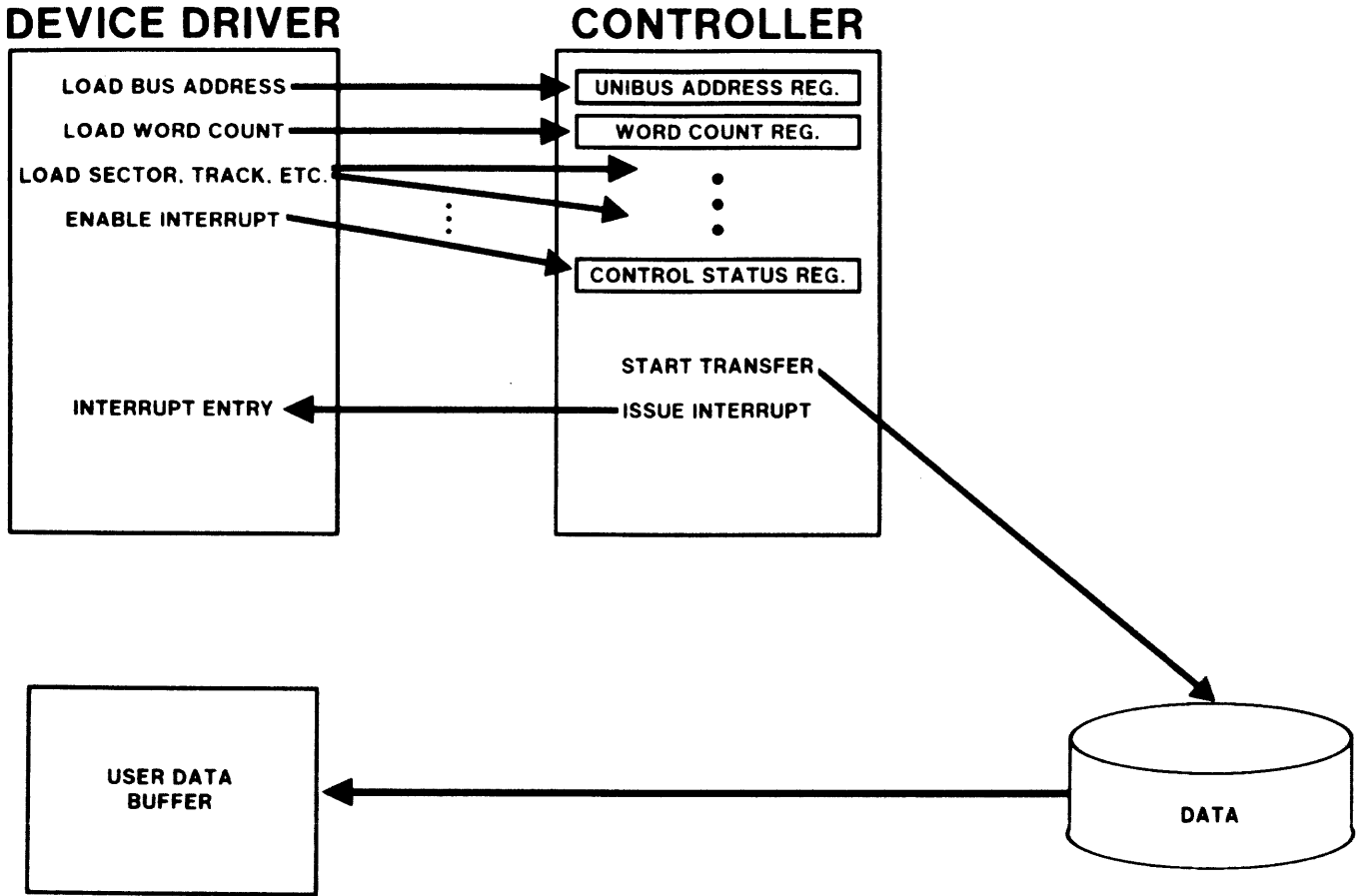Figure 13-9   Interrupt Processing - Non DMA Device

Figure 13-10   Interrupt Processing - DMA Device

ADVANCED DRIVER FEATURES

Loadable Drivers

- Not physically within the Executive, but in a user partition

- Frees part of KERNEL address space to be used to map added DSR

- LOAD command brings into memory

- UNLOAD command removes from memory

- Consists of three parts

    Interrupt Control Block in DSR

    Driver code loaded into driver partition

    Driver Data Base loaded in DSR

The Interrupt Control Block

- Created by LOAD command in DSR

- Contains

    Location to which control is transfered by interrupt

    Call to interrupt save routine, to save registers

    Code to map driver code using KERNEL APR 5

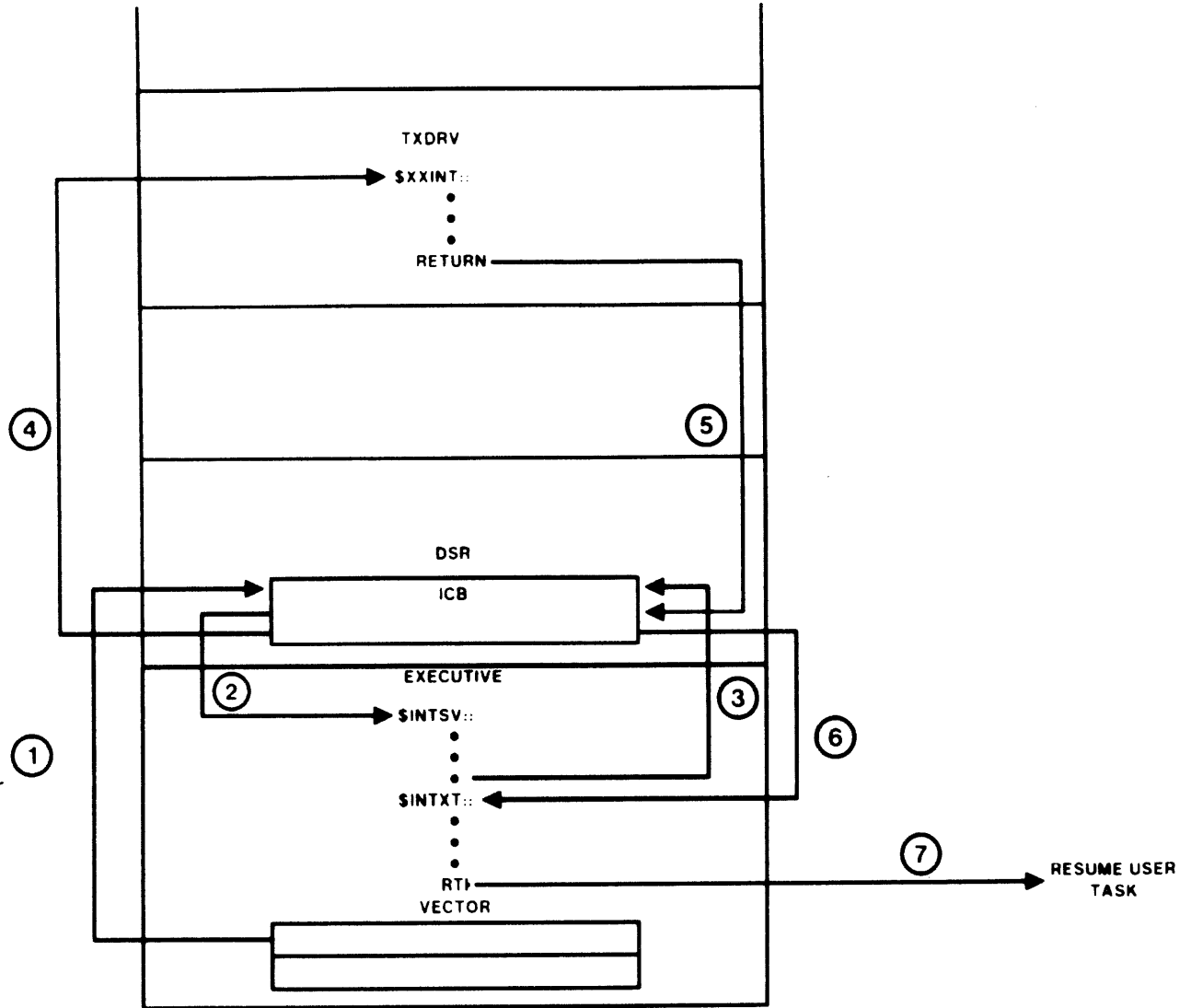Figure 13-11   A Loadable Driver

113767	
177776	MFPS TEMP
50	
004537	JSR R5, @#$INTSV
ADDR OF $INTSV	
177753(TYPICAL) .WORD ↑C <PR1>	
016704	MOV TEMP,R4[1]
000036	
043704	BIC↑C<17>,R4[1]
177760	
006304	ASL R4[1]
013746	MOV @#KISAR5, -(SP)
172352	
012737	
DRIVER RELOCATION BIAS MOV # RELBAS,@#KISAR5	
172352	
004737	JSR PC,@#INTSRV
ADDRESS OF INTERRUPT SERVICE ROUTINE	
012637	MOV (SP)+,@#KISAR5
172352	
000207	RETURN
000000	TEMP:.BLKW 1[1]

1. MULTI-CONTROLLER DRIVERS

Figure 13-12   The Interrupt Control Block

Resident and Loadable Data Base

- Data base for loadable driver can be loadable or resident

- Resident data base created when system is built and included in the Executive

- Loadable data base can be created at any time

  Loaded in DSR by LOAD command if not already in memory

  Not unloaded by UNLOAD command

UNIBUS Mapping Registers

Problem:

- Direct Memory Access (DMA) devices

  Transfer data directly between the device and physical memory

  Require physical address to be passed to UNIBUS

- In some PDP-11s, physical addresses require 22 bits

- UNIBUS accepts only 18-bit addresses

Solution:

- 31 UNIBUS Mapping Registers (UMRs) are provided

- Each UMR maps 4K words of physical memory

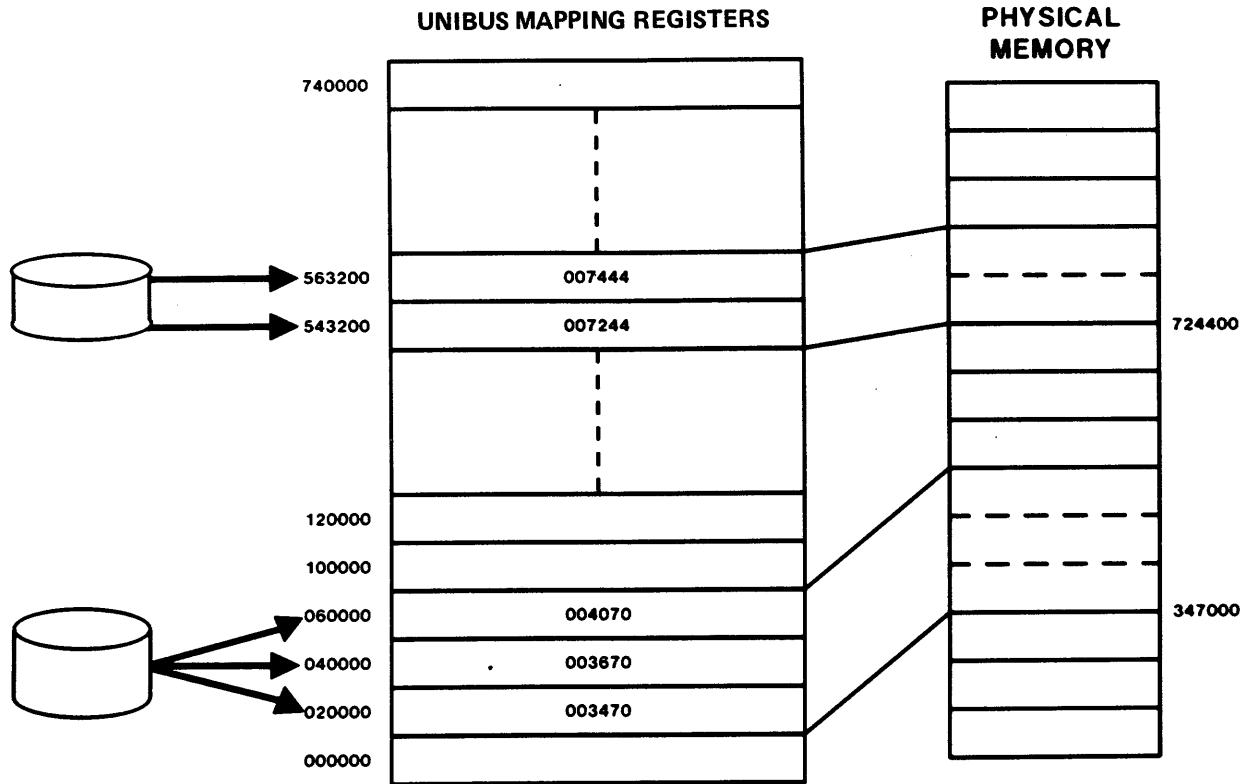- 124K words of address space just below the I/O page are relocated using UMRs

Figure 13-13   UNIBUS Mapping Registers

TRANSFER A BLOCK OF 8K WORDS

USER BUFFER ADDRESS = 1344200

UMR'S 3 AND 4 ARE TO BE USED

UMR STRUCTURE:

       BITS 13-17 = NUMBER OF UMR

       BITS 0 -12 = OFFSET TO BASE ADDRESS IN UMR

THEREFORE:

       UMR 3 = 13442

       UMR 4 = 13642

THE ADDRESS SUPPLIED TO THE UNIBUS IS 060000
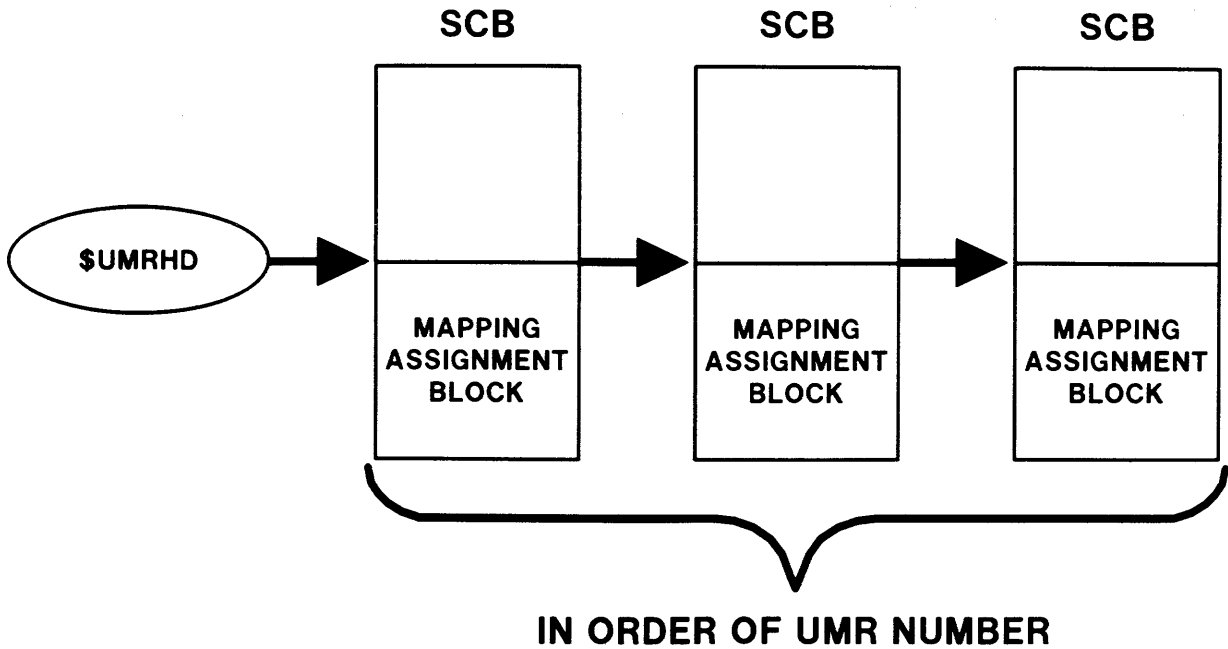
Example 13-3   UMR Address Relocation

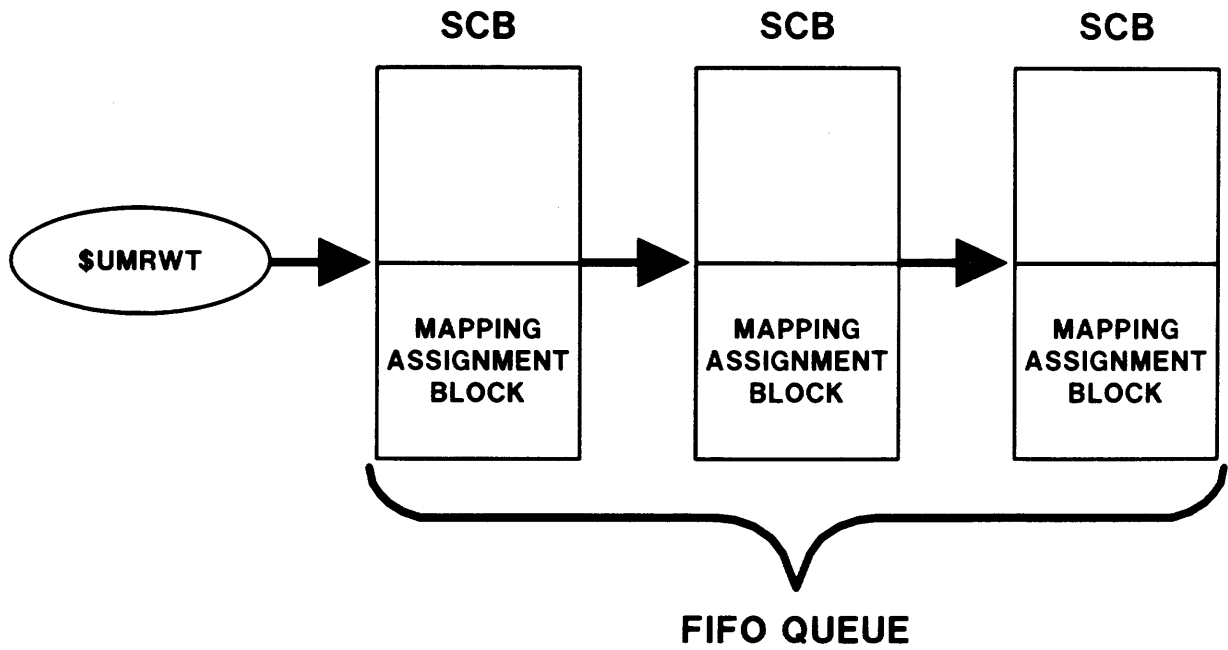Figure 13-14   Mapping Assignment Block List

Figure 13-15   UMR Wait Queue

CONNECT-TO-INTERRUPT

- Privileged task which contains interrupt service routine, ISR

    Task may or may not be mapped to Executive

- Connect-to-interrupt directive (CINT$) used to set up interrupt service

    Interrupt Transfer Block (ITB) created in pool

        Address X.JSR in ITB loaded into PC of specified vector

        Vector must be greater than 60 and unused

    ITB contains code to map the ISR through KERNEL APR 5 and transfer control to it

    ITB also contains FORK block for ISR

    ITB linked to the ITB list of the task using T.CPCB in the TCB

    Task is made noncheckpointable and nonshufflable

- Connect-to-interrupt directive performs disconnect if issued with address of ISR equal to 0

    Task is made checkpointable and shufflable regardless of original status

- Interrupt service routine is limited to 4K words
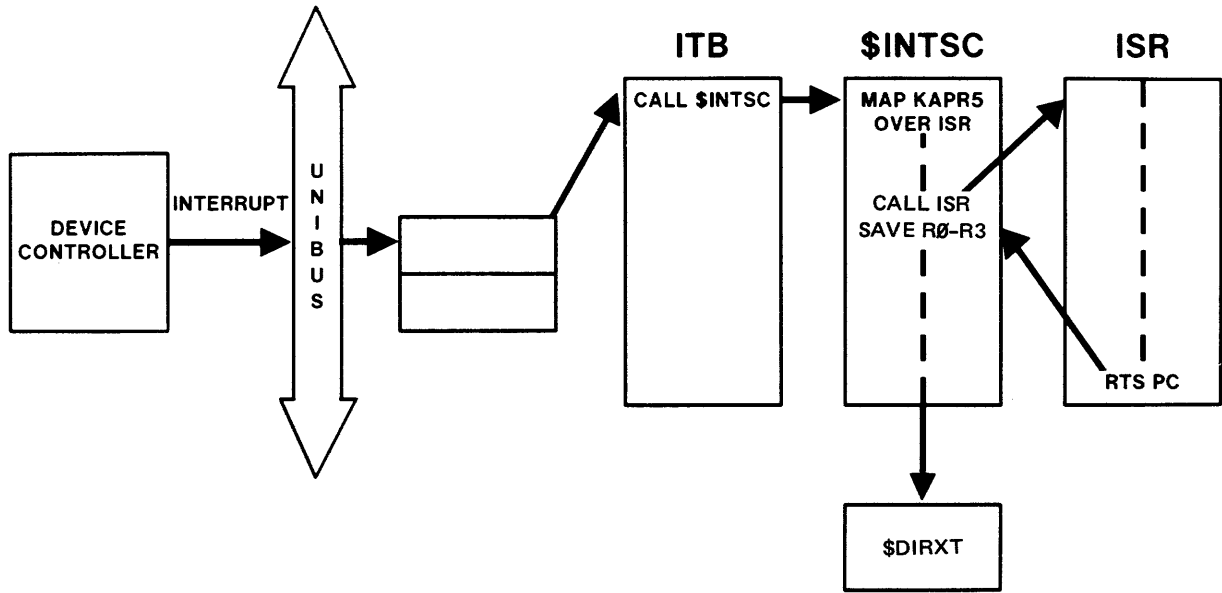
Figure 13-16   Connect-to-Interrupt Processing

LINK TO NEXT ITB	0	X.LNK

004737		2	X.JSR
ADDRESS OF $INTSC	JSR R5, #$INTSC	4	

UNUSED	LOW BYTE OF PSW FOR ISR	6	X.PSW

INTERRUPT SERVICE ROUTINE ADDRESS (APR 5 MAPPING)	10	X.ISR

FORKBLOCK	LINK TO NEXT FORK BLOCK	12	X.FORK
	SAVED PC		
	SAVED R5		
	SAVED R4		

RELOCATION BIAS FOR APR5	X.REL
ADDRESS OF DISABLE INTERRUPT ROUTINE	X.DSI
TCB ADDRESS OF OWNING TASK	X.TCB
A.DQSR FOR AST BLOCK	
AST BLOCK	X.AST
VECTOR ADDRESS	X.VEC [1]
SAVED VECTOR PC	X.VPC
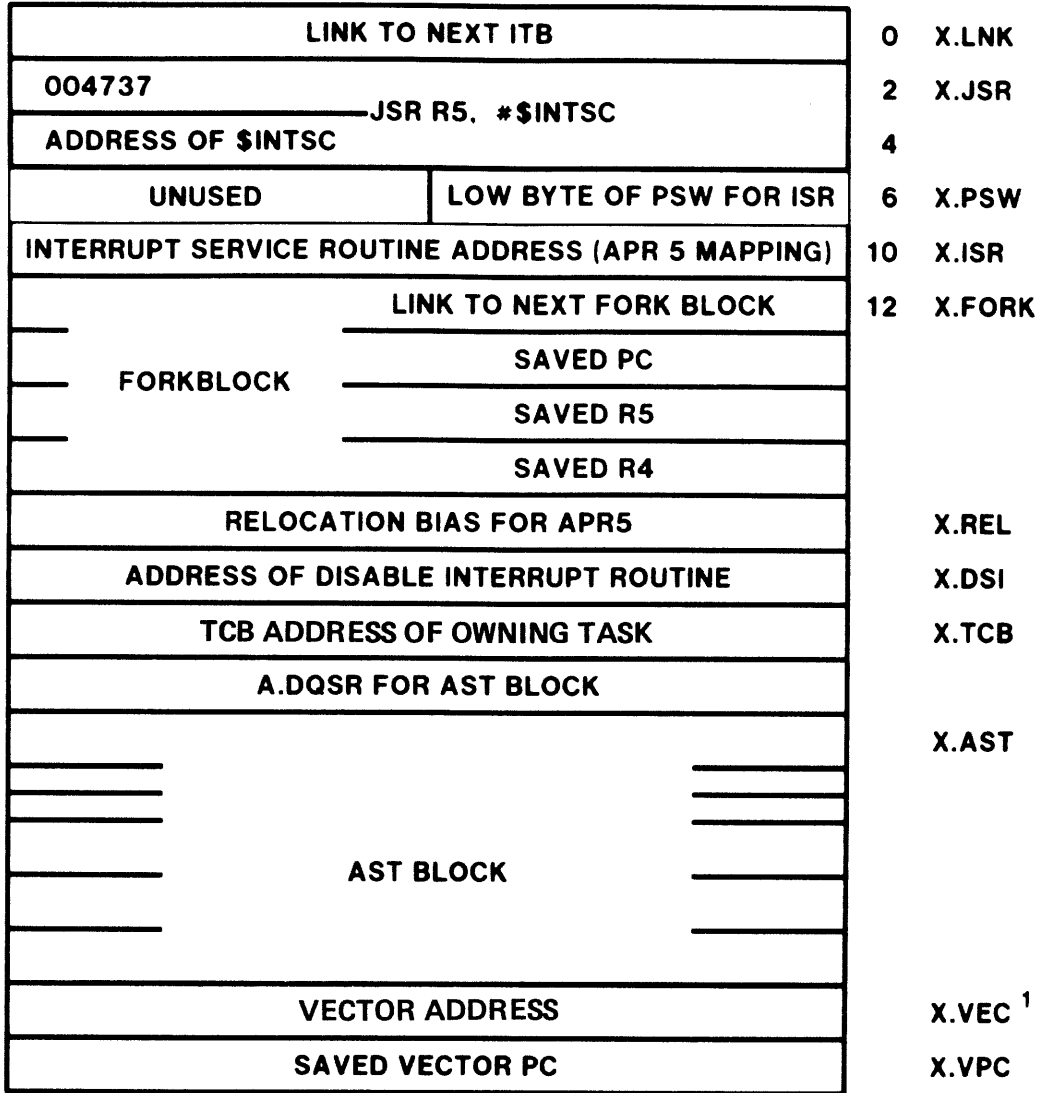
1. IF AST SUPPORT THIS IS FIRST AND ONLY AST PARAMETER

Figure 13-17   The Interrupt Transfer Block

```
        .TITLE  CTI
        .IDENT  /1.0/
;+
; ** THIS TASK ILLUSTRATES THE CONNECT TO INTERRUPT VECTOR DIRECTIVE
;       THIS PRIVILEGED TASK CONNECTS TO A DR-11K.  THERE IS A
;       LOCAL OPERATOR CONSOLE (LOC BOX) ATTACHED TO THE DR-11K
;       WHICH HAS 3 PUSH BUTTONS.  THESE 3 SWITCHES WILL CAUSE
;       THE DR-11 TO INTERRUPT AND TO PERFORM THE FOLLOWING
;       OPERATIONS:
;
;
;               SWITCH 1 - THE INTERRUPT SERVICE ROUTINE
;                          WILL SAMPLE THE TIME, AND STORE
;                          THE NUMBER OF TICKS TO THE NEXT
;                          SECOND IN A BUFFER.
;
;
;               SWITCH 2 - THE ISR QUEUES AN AST TO THIS TASK
;                          WHICH WILL AVERAGE THE VALUES COLLECTED
;                          IN THE ISR AND PRINT IT ON 'TI'.
;
;
;               SWITCH 3 - THE ISR SETS A LOCAL EVENT FLAG WHICH
;                          CAUSES THIS TASK TO DISCONNECT FROM
;                          THE VECTOR AND EXIT.
;
;
;--
        .MCALL  CINT$,EXIT$S,QIOW$,ASTX$S,WTSE$,DIR$
        .NLIST  BEX
;               DEFINITIONS FOR THE DR-11K
DRVEC   =       230       ;VECTOR
DRCSR   =       167770    ;CONTROL AND STATUS REGISTER
DRBUF   =       167772    ;INPUT BUFFER REGISTER
SWTCH1  =       1         ;SWITCH 1
SWTCH2  =       2         ;SWITCH 2
SWTCH3  =       4         ;SWITCH 3
EF      =       2         ;EVENT FLAG FOR TASK
BUFSIZ  =       30.       ;NUMBER OF SAMPLES TO STORE
;
CINT:   CINT$   DRVEC,ISRBAS,DRINT,DREDIR,PR5,DRAST      ;CONNECT
DCINT:  CINT$   DRVEC   ;DISCONNECT
WAIT:   WTSE$   EF      ;WAIT FOR EF
PRINT:  QIOW$   IO.WVB,5,1,,,,<MES,0,40>        ;PRINT AVERAGE
FORMAT: .ASCIZ  /NUMBER OF SAMPLES TAKEN %D.  AVERAGE OF THE SAMPLES %O/
ERR:    .ASCIZ  /FAILED TO CONNECT TO VECTOR.  DIRECTIVE STATUS = %D./
        .EVEN
MES:    .BLKB   100.    ;FORMATTED MESSAGE
        .ENABL  LSB
;
;       MAIN LINE CODE
;       CONNECT TO DR-11K INTERRUPT VECTOR AN PUT TASK TO SLEEP BY
;       WAITING ON LOCAL EVENT FLAG 2
;
START:
        CLR     COUNT           ;INITIALIZE NUMBER OF SAMPLES
        CLR     BUFFER          ;INITIALIZE SUM
        DIR$    #CINT           ;CONNECT TO INTERRUPT
        BCS     10$             ;FAILED TO CONNECT
        DIR$    #WAIT           ;PUT TASK TO SLEEP
        DIR$    #DCINT          ;DISCONNECT
        EXIT$S
;
;       CONNECT FAILED - PRINT ERROR MESSAGE
;
10$:    MOV     #MES,R0         ;OUTPUT BUFFER
        MOV     #ERR,R1         ;FORMAT SPECIFICATION
```

Example 13-4 Sample Connect-to-Interrupt Routine
(Sheet 1 of 3)

```
        MOV     #$DSW,R2            ;VALUE TO BE CONVERTED
        CALL    $EDMSG             ;FORMAT MESSAGE
        MOV  ·  R1,PRINT+Q.IOPL+2 ;NUMBER OF BYTES IN MESSAGE
        DIR$    #PRINT             ;PRINT ERROR MESSAGE
        EXIT$S
;
;       AST WILL AVERAGE THE TIME SAMPLES AND PRINT ON TI
;
DRAST:
        MOV     COUNT,R0           ;NUMBER OF SAMPLES
        BEQ     35$                ;SKIP AVERAGING
        CLR     R1                 ;ACCUMULATE SUM IN R1
        MOV     #BUFFER,R2         ;SAMPLES TO SUM
15$:    MOVB    (R2)+,R3           ;GET SAMPLE
        BIC     #177400,R3         ;DON'T SIGN EXTEND
        ADD     R3,R1              ;ADD INTO TOTAL
        SOB     R0,15$             ;GET NEXT VALUE
        DIV     COUNT,R0           ;GET AVERAGE
        MOV     R1,BUFFER          ;STORE QUOTIENT
35$:    MOV     #MES,R0            ;OUTPUT STRING
        MOV     #FORMAT            ;FORMAT SPECIFICATION
        MOV     #COUNT,R2          ;NUMBERS TO CONVERT
        CALL    $EDMSG             ;CONVERT MESSAGE
        MOV     R1,PRINT+Q.IOPL+2 ;NUMBER OF BYTES TO PRINT
        DIR$    #PRINT             ;WRITE SAMPLE AVERAGE
        TST     (SP)+              ;POP VECTOR ADDRESS
        CLR     BUFFER             ;RE-INITIALIZE SUM
        CLR     COUNT              ;AND COUNT OF SAMPLES
        ASTX$S                     ;EXIT AST
;
;       ISRBAS DEFINES THE BASE ADDRESS OF A 4K-WORD AREA THE
;       EXECUTIVE WILL MAP WITH KERNEL APR5.   THIS AREA MUST
;       CONTAIN THE INTERRUPT SERVICE ROUTINE, THE ENABLE/DISABLE
;       INTERRUPT ROUTINE, AND ANY DATA AREAS THAT THESE ROUTINES
;       WILL REFERENCE.
;
ISRBAS:
;       DATA BUFFERS NEEDED BY ISR
COUNT:  .WORD   0                  ;NUMBER OF SAMPLES
BUFFER: .BLKB   BUFSIZ             ;BUFFER FOR INPUTS
        .EVEN
;
;       ENABLE/DISABLE INTERRUPTS ROUTINE
;       AUTOMATICALLY CALLED UNDER 3 CONDITIONS
;               1.  WHEN THE TASK CONNECTS TO AN INTERRUPT VECTOR
;                   TO ENABLE INTERRUPTS.
;               2.  WHEN THE TASK DISCONNECTS FROM THE INTERRUPT
;                   VECTOR TO DISABLE INTERRUPTS.
;               3.  WHEN THE TASK IS ABORTED TO DISABLE INTERRUPTS.
;
DREDIR: BCS     20$                ;IF CS DISABLE INTERRUPTS
        BIS     #100,@#DRCSR       ;ENABLE INTERRUPTS
        RETURN                     ;AND OUT
20$:    BIC     #100,@#DRCSR       ;DISABLE INTERRUPTS
        RETURN                     ;AND OUT
;
;       INTERRUPT SERVICE ROUTINE ·
;
;       THE ISR IS ENTERED AT PRIORITY 5 (SPECIFIED ON CINT$) WITH
;               R4 = SAVED AND FREE TO BE USED
;               R5 = ADDRESS OF FORK BLOCK IN THE ITB
;
DRINT:  BIT     #SWTCH1,@#DRBUF  ;;;INPUT BUFFER REQUEST?
```

Example 13-4   Sample Connect-to-Interrupt Routine
(Sheet 2 of 3)

```
        BEQ     30$             ;;;IF EQ, NO
        CMP     #BUFSIZ,COUNT   ;;;YES, BUFFER FULL?
        BEQ     50$             ;;;IF EQ, YES
        MOV     COUNT,R4        ;;;IF NE, NO
        ADD     PC,R4           ;;;CALCULATE BUFFER
        ADD     #BUFFER-.,R4    ;;;PIC ADDRESS TO STORE SAMPLE
        MOVB    @#$TTNS,(R4)    ;;;LOW BYTE OF TICKS TILL NEXT SECOND
        INC     COUNT           ;;;INCREMENT NUMBER OF SAMPLES
        BR      50$             ;;;EXIT FROM INTERRUPT
;
;       INTERRUPT WAS FROM SWITCH 2 OR 3
;
30$:    CALL    @#$FORK2        ;;;CREATE A SYSTEM PROCESS
        CLR     (R3)            ;DECLARE THE FORK BLOCK FREE
        BIT     #SWTCH3,@#DRBUF  ;DISCONNECT FROM INTERRUPT?
        BNE     40$             ;IF NE, YES
;
;       SWITCH 2 => QUEUE AN AST TO THE TASK
;
        CALL    @#$QASTC        ;QUEUE AN AST
        BR      50$             ;EXIT FROM INTERRUPT
;
;       SWITCH 3 => DISCONNECT FROM INTERRUPTS.
;       SET LOCAL EVENT FLAG 2 TO WAKE UP TASK LEVEL CODE TO
;       ISSUE THE DISCONNECT
;
40$:    MOV     #EF,R0          ;EVENT FLAG 2 TO BE SET
        MOV     X.TCB-X.FORK(R5),R5 ;R5 POINTS TO THE TCB OF TASK
        CALL    @#$SETF         ;EXEC SUBROUTINE SETF
        RETURN                  ;EXIT INTERRUPT
;
;       EXIT FROM INTERRUPT.  TURN INTERRUPT BACK ON.
;
50$:    BIS     #100,@#DRCSR    ;TURN ON INTERRUPTS
        RETURN                  ;EXIT FROM INTERRUPT
;
;       ENSURE ISR + DATA + ENABLE/DISABLE SUBROUTINES FIT IN 4K
;
        .IIF    LT,<20000 - <. - ISRBAS>> .ERROR
        .END    START
```

Example 13-4   Sample Connect-to-Interrupt Routine
(Sheet 3 of 3)