RSX–11M

Guide to Writing an I/O Driver

Order No. AA-2600E-TC

digital

software

RSX–11M

Guide to Writing an I/O Driver

Order No. AA-2600E-TC

RSX–11M Version 4.0

digital equipment corporation · maynard, massachusetts

First Printing, April 1975
Revised, September 1975
Revised, November 1976
Revised, December 1977
Updated, May 1979
Revised, November 1981

Printed in U.S.A.

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

ZK2048

CONTENTS

CONTENTS

Page

CONTENTS

Page

CONTENTS

EXAMPLES

FIGURES

TABLES

PREFACE

MANUAL OBJECTIVES

The goal of this manual is to provide information necessary to prepare a conventional I/O driver for RSX-11M and subsequently incorporate it into an operational user-tailored system. A "conventional" driver is best explained by example. Disks and unit record devices are considered conventional; the LPS-11, UDC-11, and TM11 are considered unconventional. Complexity of device servicing requirements sets the dividing line, which is not easily established in black-and-white terms.

INTENDED AUDIENCE

The manual assumes that you understand the device for which you are writing a driver, and that you are familiar with the PDP-11 processor, its peripheral devices, and the software supplied with an RSX-11M system. Although this manual is organized tutorially, the intended audience is assumed to be at a system programmer level of expertise; thus, the manual does not contain definitions of data processing terms and concepts familiar to senior-level professionals.

STRUCTURE OF THIS DOCUMENT

This document proceeds from chapter to chapter toward increasingly detailed levels of implementation.

Chapter 1 is a general introduction to I/O drivers in the RSX-11M system.

Chapter 2 is a functional description of the RSX-11M device-level I/C system. It discusses both data structure and code requirements.

Chapter 3 details how to incorporate a user-written driver into the system.

Together, Chapters 1, 2, and 3 provide a complete description of the requirements that must be met in creating a system that contains a user-written driver.

Chapter 4 provides programming-level details on I/O data structures and on drivers that service several controllers.

Chapter 5 discusses all the I/O-related Executive services.

Chapter 6 gives two examples of user-written drivers.

Appendix A describes the address doubleword.

Appendix B outlines special considerations for extended memory NPR device drivers.

Appendix C lists system macros that supply symbolic offsets for system data structures.

Appendix D is a guide for the user in developing an Ancillary Control Processor (ACP).

ASSOCIATED DOCUMENTS

Familiarity with the system implies an in-depth exposure to the following manuals:

- RSX-11M System Generation and Installation Guide
- RSX-11M/M-PLUS I/O Drivers Reference Manual
- RSX-11M/M-PLUS Executive Reference Manual
- RSX-11M/M-PLUS Utilities Manual
- IAS/RSX-11 I/O Operations Reference Manual

As adjuncts to this manual, you are advised to study existing I/O drivers. The RL-11 disk driver is a good example of an NPR device and the TA-11 (cassette) is illustrative of a programmed I/O device. In addition, a perusal of Executive source code contained in the files IOSUB, SYSXT, DRQIO, BFCTL, and DRDSP (stored in UFD [11,10] on the Executive source disk) is recommended.

Other manuals closely allied to the purposes of this document are described briefly in the RSX-11M/RSX-11S Information Directory and Index. The Information Directory defines the intended readership of each manual in the RSX-11M/RSX-11S set and provides a brief synopsis of each manual's contents.

SUMMARY OF TECHNICAL CHANGES

This revision of the RSX-11M Guide to Writing an I/O Driver incorporates the following technical changes and additions:

1. Tables have been added to aid the user in establishing I/O function bit masks for disk drives, magtape drives, and unit record devices (see Tables 4-3, 4-4, and 4-5).

2. Error logging offsets have been added to the SCB and UCB. See the descriptions of the SCB and the UCB in Chapter 4.

3. Other additions have been made to various UCB offsets:

   - New symbolic name U.MUP has been added (redefinition of U.CLI; see Chapter 4).

   - Symbolic names for magtape density bit masks have been added to the description of U.CW3 (see Chapter 4).

   - The DV.MBC offset to U.CW1 has been renamed to DV.EXT (see Chapter 4).

4. Some Executive routines listed in Chapter 5 have been moved to new Executive modules. The following is a list of the affected modules and subroutines:

Routine	Old Module	New Module
$ACHKB/$ACHCK	IOSUB	EXESB
$ASUMR	IOSUB	MEMAP
$DEUMR	IOSUB	MEMAP
$DVMSG	IOSUB	EXESB
$MPUBM	IOSUB	MEMAP
$MPUB1	IOSUB	MEMAP
$RELOC	IOSUB	MEMAP
$STMAP	IOSUB	MEMAP
$STMP1	IOSUB	MEMAP

   In addition, the inputs to $MPUB1 have been modified (see Chapter 5).

5. Three additional routines have been documented in Chapter 5:

   $EXRQP, $QRMVF, and $SWSTK.

6. An appendix intended to aid the user in writing an Ancillary Control Processor (ACP) has been added (see Appendix D).

CHAPTER 1

INTRODUCTION TO I/O DRIVERS

The software supplied by DIGITAL for an RSX-11M system includes I/O drivers for a number of standard I/O devices. An I/O driver is a part of the RSX-11M Executive that services one type of I/O device. A driver may handle one or several controllers, each with one or several device-units attached.

1.1 RESIDENT AND LOADABLE DRIVERS

A driver can be resident or loadable. A resident driver is a permanent part of the Executive, incorporated at system generation. A loadable driver, while also part of the Executive, can be added to or unloaded from a system almost at will by means of MCR or VMR commands. During the system generation dialog, you can specify that you want this facility.

Making drivers loadable can result in less Executive code, and thus permits an increase in available dynamic storage region (pool) space or increased space for user tasks. You can unload from the system any driver that is not needed for a period of time. For example, assume your system has both a paper tape reader and a card reader. If only one of them is connected to the system at any one time, you could load the driver for the on-line device and unload the other driver, thus reducing the size of the Executive.

A loadable user-written driver is easier to incorporate into a system and easier to debug than a resident one. You can incorporate a resident driver into a system only during system generation; the Executive must be rebuilt and the system bootstrapped each time it is incorporated after debugging. In contrast, you can incorporate a loadable driver into a system with a single MCR command. Incorporating and debugging user-written drivers are discussed in Chapter 3.

1.2 FUNCTION OF AN I/O DRIVER

An I/O driver is an asynchronous process (not a task) that calls and is called by the Executive to service an external I/O device or devices. The role of an I/O driver in the RSX-11M I/O structure is specific and limited. A driver performs the following functions:

- Receives and services interrupts from its I/O device(s)
- Initiates I/O operations when requested to do so by the Executive
- Cancels in-progress I/O operations
- Performs other (device-specific) functions upon power failure and device time-out

As an integral part of the Executive, a driver possesses its own context, allows or disallows interrupts, and synchronizes its access to shared data bases with that of other Executive processes. It may also synchronize with itself: A driver can handle several device controllers (each with several device-units) all operating in parallel. A user-written driver must adhere to RSX-11M programming conventions in order to ensure the integrity of the Executive. Section 2.5 and Chapter 4 discuss these conventions.

CHAPTER 2

THE RSX-11M I/O SYSTEM--PHILOSOPHY AND STRUCTURE

2.1 I/O PHILOSOPHY

Memory constraints and compatibility requirements dominated the design philosophy and strategy used in creating RSX-11M. To meet its performance and space goals, the RSX-11M I/O system attempts to centralize common functions, thus eliminating the inclusion of repetitive code in each and every driver in the system. To achieve this centralization, a substantial effort has been expended in designing the RSX-11M data structures, which are used to drive the centralized routines. The effect is to reduce substantially the size of individual I/O drivers. The table structures require space and must be considered with the total size of the I/O system. Nevertheless, the size reduction effected by centralizing functions, combined with table-driven design, enables RSX-11M to meet its intended memory and performance goals.

2.2 STRUCTURE

The following sections:

1. Place an I/O driver in the context of the overall RSX-11M I/O system
2. Establish the responsibilities of an I/O driver
3. Functionally describe the driver's interface to the Executive subroutines and the I/O data structures

2.2.1 I/O Hierarchy

The RSX-11M I/O system is structured as a loose hierarchy. The term "loose" indicates that you can enter the hierarchy at any of its levels; this characteristic is contrasted to "tight" hierarchies that permit entry only from the top level. Tight hierarchies exist principally for security and protection, but consume equipment resources. Figure 2-1 shows the loose RSX-11M I/O system hierarchy.

2.2.1.1 **FCS/RMS** - At the top of the hierarchy are File Control Services (FCS) and Record Management Services (RMS), which provide device-independent access to devices included in a given system configuration. The user task has two levels with which to interface with FCS or RMS: Get/Put and Read/Write. Get/Put facilitates the movement of data records, whereas Read/Write provides a more basic level of access affording more direct control over data movement between a task and peripheral devices.

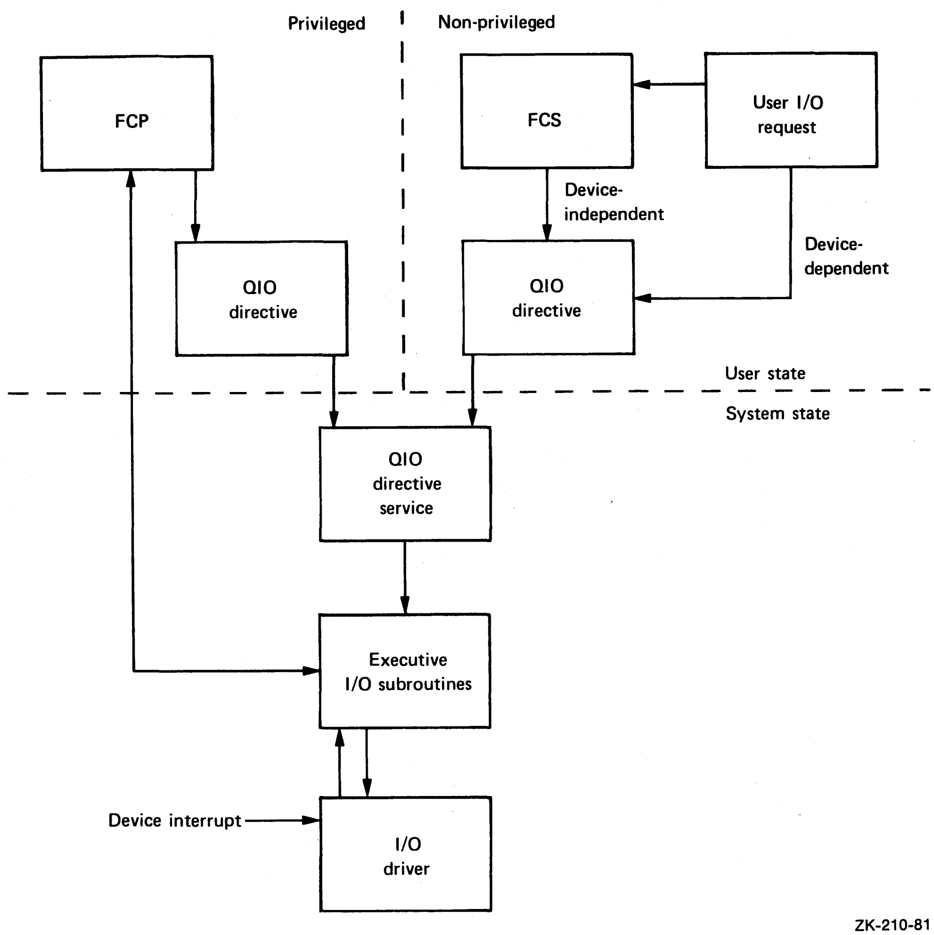


Figure 2-1 I/O Control Flow

The discussion of FCS and RMS is brief because their existence is completely transparent to the driver and rarely concerns the writer of a conventional driver. FCS or RMS accepts a user request, interprets it, and translates it into a series of low-level system directives known as QIOs.

2.2.1.2 QIO - The QIO directive is the lowest level of task I/O. Any task can issue a QIO directive which allows direct control over devices that are connected to a system and that have an I/O driver. The QIO directive forces all I/O requests from user tasks to go through the Executive. The Executive works to prevent tasks from destructively interfering with each other and with the Executive itself.

2.2.1.3 **Executive I/O Processing** - The Executive processes I/O requests using the following:

1. An Ancillary Control Processor (ACP)

2. A collection of Executive components consisting of:

   a. QIO directive processing

   b. I/O-related subroutines

   c. The I/O drivers

An ACP is responsible for maintaining the structure and integrity of data stored on file-structured volumes. It maps virtual I/O functions to logical I/O functions, and makes volume-protection checks. The driver converts a logical block number into a physical address on a file-structured device. No direct connection normally exists between the ACP and a driver.

An ACP is a privileged task, and requires a partition in which to execute. Drivers, on the other hand, are specialized system processes, not tasks.

The Executive provides QIO directive processing. It also provides a collection of subroutines that are used by drivers to obtain I/O requests, to facilitate interrupt handling, and to return directive status codes. Actual control of the device is performed by the driver. These subroutines are examined in detail in Chapter 5. Executive subroutine services and QIO directive processing are shown as distinct components but are, in fact, both part of the Executive. These subroutines centralize common driver functions, thus eliminating duplicate code sequences among drivers.

2.2.2 Role of I/O Driver in RSX-11M

Every I/O driver in the RSX-11M system has the following five entry points:

- Device interrupt[1]
- I/O initiator
- Device time-out
- Cancel I/O
- Power failure

The first entry point is entered by a hardware interrupt; the other four are entered by calls from the Executive. Functional details regarding these entry points follow.

---

1. A device may trigger more than one distinct interrupt entry. For example, a full-duplex device would have two.

2.2.2.1 **Device Interrupt** - Control passes to this entry point when a device previously initiated by the driver completes an I/O operation and causes an interrupt in the central processor. The connection between the device and the driver in this instance is direct; the Executive is not involved.

2.2.2.2 **I/O Initiator** - The Executive calls this entry point to inform the driver that work for it is waiting to be done. In effect, this is a wake-up signal for the driver.

2.2.2.3 **Device Time-out** - When a driver initiates an I/O operation, it can establish a time-out count. If the function then fails to complete within the specified time interval, the Executive notes the time lapse and calls the driver at this entry point.

2.2.2.4 **Cancel I/O** - A number of circumstances arise within the system that require a driver to terminate an in-progress I/O operation. When such a termination becomes necessary, a task so informs the Executive, which then relays the request to the driver by calling it at the cancel I/O entry point.

2.2.2.5 **Power Failure** - The Executive calls the driver's power-failure entry point in three different circumstances:

- When power is restored after a failure
- When the system is bootstrapped
- When the driver is loaded (if it is a loadable, as opposed to a resident, driver)

Power Restore - Two conditions can initiate a call to the driver when power is restored following a power failure. First, the Executive automatically calls the power-failure entry point when power is restored any time the controller is busy (that is, when I/O is in progress). Second, a driver has the option to be called regardless of the existence of an outstanding I/O operation at the time the power is restored (See the description of the UC.PWF bit symbol in Section 4.1.4.1). Frequently, a driver's response to a power failure or to an I/O failure is identical to that when its device times out; in such a case, the power-failure entry point may simply be a RETURN instruction, because the driver will recover eventually via the time-out entry point.

Bootstrap - When the system is bootstrapped, a power-failure interrupt is simulated. This simulation gives drivers the opportunity to carry out any appropriate preoperational initialization.

Load - The MCR LOA command calls a loadable driver at its power-failure entry point if the device is on line and UC.PWF is set (see Section 4.1.4.1).

2.2.2.6 **Summary--Role of an I/O Driver** - Functionally, a driver in RSX-11M is responsible for:

- Servicing device interrupts
- Initiating I/O operations
- Cancelling in-progress I/O operations
- Performing device-related functions when a time-out or power failure occurs

A driver exists as an integral part of the Executive; it can call, and be called by, the Executive. The driver initiates device I/O operations directly and receives device interrupts directly. All other entry points are entered by means of Executive calls. A driver never receives a QIO request directly, and has no direct interaction with the ACP.

2.3 DATA STRUCTURES

An I/O driver interacts with the following data structures:

- Device Control Blocks (DCBs)
- Unit Control Blocks (UCBs)
- Status Control Blocks (SCBs)
- The I/O packet
- The I/O queue
- The fork list
- Device interrupt vector

The first four of these data structures are especially important to the driver, because it is by means of these data structures that all I/O operations are effected. They also serve as communication and coordination vehicles between the Executive and individual drivers.

The I/O queue and the fork list are actually Executive data structures, but are discussed here to illustrate all facets of the interaction of an I/O driver with the Executive. The I/O queue is a list of I/O packets built by the QIO directive, principally from information in the caller's Directive Parameter Block (DPB). The fork list synchronizes access to shared Executive data structures.

Entry to a driver following a device interrupt is accomplished through the appropriate hardware device interrupt vector. As will be seen, writers of resident drivers are responsible for properly initializing this vector. It is discussed in conjunction with the data structures associated with a driver.

2.3.1 The Device Control Block (DCB)

At least one DCB exists for each type of device appearing in a system (device type should not be equated with device-unit). The function of the DCB is to describe the static characteristics (rather than execution-time information) of both the device controller and the units attached to the controller. All the DCBs in a system form a forward-linked list, with the last DCB having a link of zero. Most of the data in the DCB is established in the assembly source for the driver data structure. The DCB is used by the QIO directive processing code in the Executive and not by the driver.

2.3.2 The Unit Control Block (UCB)

One UCB exists for each device-unit attached to a system. Much of the information in the UCB is static, though a few dynamic parameters exist.[1] For example, the redirect pointer, which reflects the results of the MCR RED command is updated dynamically. As with the DCB, most of the UCB is established in the assembly source; however, its contents are used and modified by both the Executive and the driver. Usually, either the Executive or the driver -- but not both -- modify a datum.

2.3.3 The Status Control Block (SCB)

One SCB exists for each device controller in the system. This is true even if the controller handles more than one device-unit (like the RL11 Controller). Line multiplexers such as the DH11 and DJ11 are considered to have one controller for each line because all lines can transfer in parallel. Most of the information in the SCB is dynamic. Both the Executive and the driver use the SCB.

2.3.3.1 **Interrelation of the I/O Control Blocks** - This section discusses the interrelationship among the DCB, UCB, and SCB without entering into the detailed contents of the control blocks, which are discussed in Chapter 4.

Figure 2-2 shows the data structure that would result for three LA36 DECwriters interfaced by means of a DH11 multiplexer. The structure requires one DCB, three UCBs, and three SCBs, because the activity on all three units can proceed in parallel.

Figure 2-3 depicts the internal data structure for an RK11 disk controller with three units attached. Note that only one SCB exists, because only one of the three units can be active at any given time.

---

1. From the UCB, however, it is possible to access most of the other structures in the I/O data base (see Figure 2-5). In this sense the UCB gives access to a large amount of dynamic data.

Figure 2-4 shows the data structure for two RK11 disk controllers, each of which has two drives attached. Here there are two SCBs, because both of the disk controllers can operate in parallel.

Taken together, Figures 2-2, 2-3, and 2-4 illustrate the strategy underlying the existence of three basic I/O control blocks. There need be only one DCB for each device type. There may be one or more SCBs, depending on the degree of parallelism that is desired or possible: one for each device-unit, or one for each controller servicing several device-units. The number of UCBs and SCBs, and their interrelationships, are uniquely determined by the hardware these data structures describe.

This data structure provides considerable flexibility in configuring I/O devices, and, because of the information density in the structures themselves, substantially reduces the code requirements of the associated drivers.

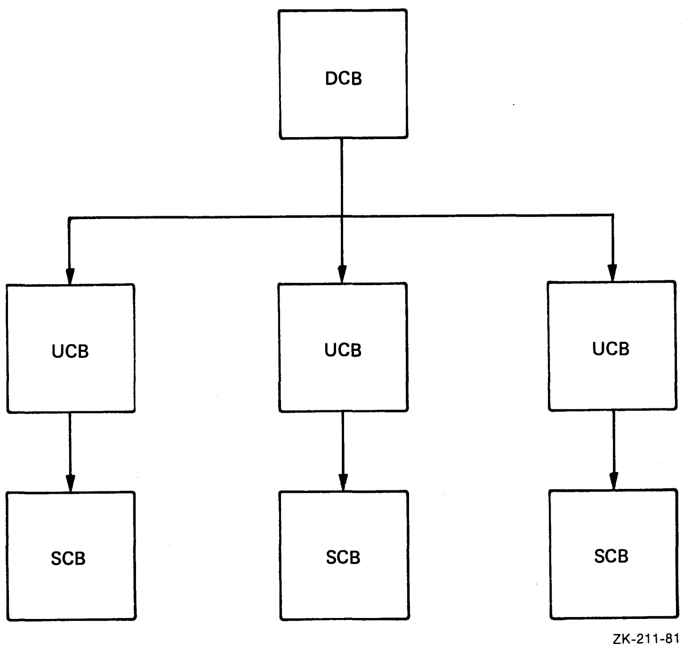


Figure 2-2 DH11 Terminal I/O Data Structure

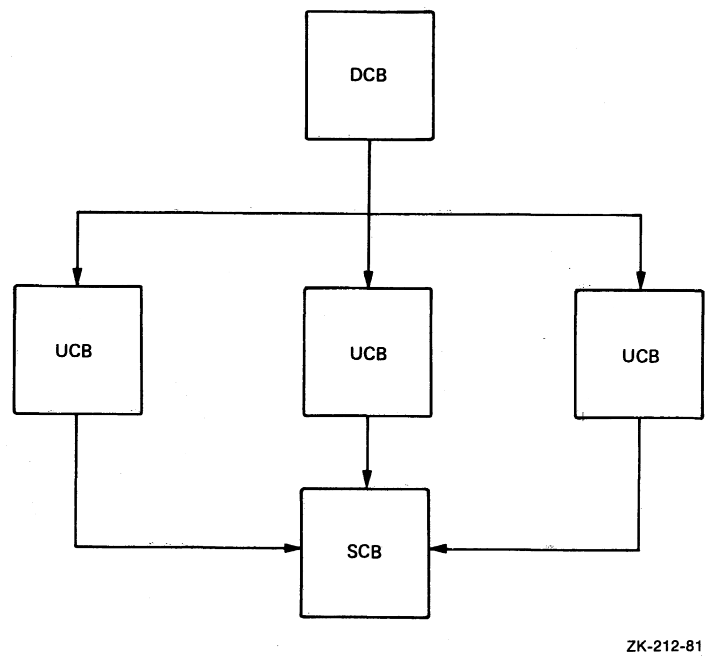


Figure 2-3 RK11 Disk I/O Data Structure

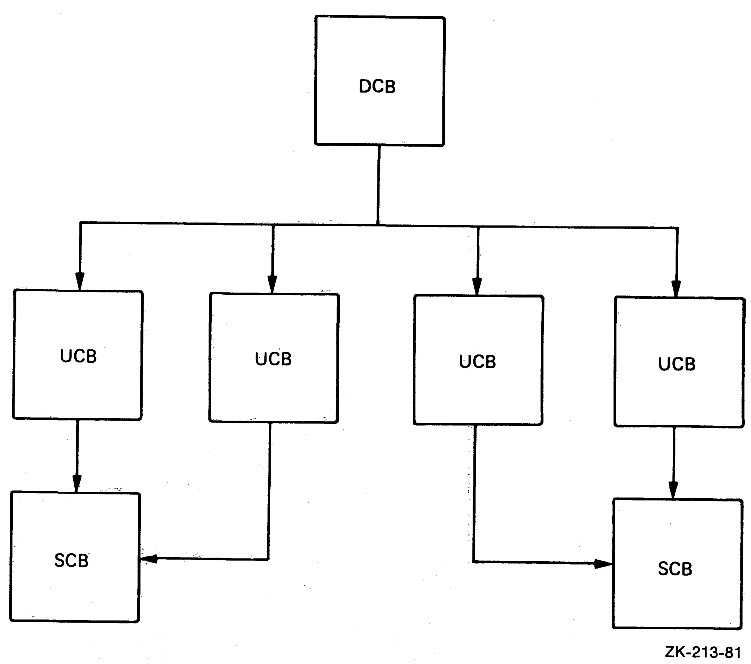


Figure 2-4 I/O Data Structure for Two RK11 Disk Controllers

2.3.4 The I/O Packet

An I/O packet contains information extracted from the QIO DPB, as well as other information needed to initiate and terminate I/O requests.

2.3.5 The I/O Queue

Each time a task makes an I/O request, the Executive performs a series of validity checks on the DPB parameters. If these checks prove successful, the Executive generates a data structure called an I/O packet. The Executive then inserts the packet into a device-specific, priority-ordered list of packets called the I/O queue. Each I/O queue's listhead is located in the SCB to which the I/O requests apply.

When a device driver needs work, it requests the Executive to dequeue the next I/O packet and deliver it to the requesting driver. Normally, the driver does not directly manipulate the I/O queue.[1]

2.3.6 The Fork List

The fork list is a mechanism by which RSX-11M splits off processes that require access to shared data bases, or that require more CPU time to process an interrupt than is compatible with fast, real-time response of the overall system.

A process that calls $FORK requests the Executive to transform it into a "fork process" and place it on the fork list. What this means is that a call to $FORK saves a "snapshot" of the process (registers R4, R5, and the PC) in a fork block. This fork block is queued on the fork list in first-in-first-out (FIFO) order.

When a fork process has worked its way to the front of the fork list, R4 and R5 are restored and execution restarts at the statement after CALL $FORK. The process is unaware that a pause of indeterminate length has occurred.

A fork process exists in a status intermediate between an interrupting routine and an ordinary task requesting system resources. Routines in the system stack--requests for interrupt processing--are run first. They can be interrupted only by higher-priority requests. Routines in the fork list are run when the system stack is empty--that is, they are completely interruptable. Finally, other tasks are run only when both the system stack and the fork list are empty.

Driver interrupts are processed at priority 7 and are thus noninterruptable. By system convention, no process should run in a noninterruptable mode for more than 100 microseconds. This convention ensures prompt attention to interrupting real-time events.

In practice, drivers servicing interrupts drop from priority 7 to a lower priority (namely, that of the interrupting source) after issuing a few instructions. Another system convention states that processing

---

1. An exception is the case in which a driver needs to examine an I/O packet before it is queued, or in place of having it queued. For the driver to accomplish this examination, you must set the bit UC.QUE in the control byte (U.CTL) of the UCB (described in Section 4.1.4).

The most common reason for a driver to examine a packet before queuing is that the driver employs a special user buffer, other than the normal buffer used in a transfer request. Within the context of the requesting task, the driver must address-check and relocate such a special buffer. See Section 6.3 for an example of a driver that does this.

at this partially interruptable level should not exceed 500 microseconds. Often, more time than this is required to process an interrupt. The fork list mechanism provides a secondary interrupt stack whose members are processed first-in-first-out whenever the system stack is empty.

A process can also call $FORK to access a shared data base--a system table, for example. You must strictly control such access in order to avoid conflicts. Under RSX-11M, many drivers can run in parallel; a multicontroller driver can run in parallel with itself. In these circumstances access to common data bases must be controlled.

Of the two available methods of controlling such access--interrupt lockout and priority queuing--RSX-11M uses priority queuing. The fork list is the queue of requests for such access. Fork processes are granted FIFO access to common data bases. Once granted such access, a process is guaranteed control of the data base until it exits.

2.3.7 The Device Interrupt Vector

The device interrupt vector consists of two consecutive words giving the address of the interrupt-service routine and the priority at which it is to run (always set to PR7). The low four bits of the second word of the interrupt vector must contain the number of the controller that interrupts through the vector. This requirement enables a driver to service several controllers with few code changes (see Sections 4.2 and 4.3 for an example and discussion of multicontroller driver coding). Generally, these bits are set to 0.

2.4 EXECUTIVE SERVICES

The Executive provides services related to I/O drivers that can be categorized as pre- and post-driver initiation. The preinitiation services are those performed by the Executive during its processing of a QIO directive; these services are not available as Executive calls.

The goal of pre-driver initiation processing is to extract from the QIO directive all I/O support functions not directly related to the actual issuance of a function request to a device. If the outcome of pre-driver initiation processing does not result in the queuing of an I/O packet to a driver, the driver is unaware that a QIO directive was issued. Many QIO directives do not result in the initiation of an I/O operation.

The post initiation services are those available to the driver after it has been given control, either by the Executive or as the result of an interrupt. They are available as needed by means of Executive calls.

An important concept used in this section and in Section 2.5 is the "state" of a process. In RSX-11M, a process can run in one of two states: user or system. Drivers operate almost entirely in the system state; the programming standards described in Section 2.5 apply to system-state processes.

2.4.1 Pre-Driver Initiation Processing

In processing a QIO directive, the Executive module DRQIO performs the following pre-driver initiation services:

1. Checks to verify whether the supplied logical unit number (LUN) is legal. If not, the directive is rejected.

2. If the LUN is valid, checks to verify whether a valid UCB pointer exists in the Logical Unit Table (LUT) for the specified LUN. This test determines if the LUN is assigned. If the test fails, the directive is rejected.

3. If steps 1 and 2 are successful, traces down the redirect chain to locate the correct UCB to which the I/O operation will actually be directed.

4. Checks the event flag number (EFN) and the address of the I/O Status Block (IOSB). If either is illegal, the directive is rejected. Immediately following validation, the Executive resets the subject event flag and clears the IOSB.

5. Obtains 18 words of dynamic storage and builds the device-independent portion of an I/O packet.

   If steps 1 through 5 succeed, the Executive sets the directive status to +1.

   Note that directive rejections in any of the above steps are completely transparent to the driver. Such rejections cause a return of carry bit set.

6. Checks the validity of the function being requested and, if appropriate, checks the buffer address, byte count, and alignment requirements for the specified device.

   If any of these checks fails, the Executive calls the I/O Finish routine ($IOFIN). $IOFIN sets the I/O status and clears the QIO request from the system.

7. If the requested function does not require a call to the driver, the Executive takes the appropriate actions and calls $IOFIN.

8. If all I/O packet checks are positive, the Executive places the I/O packet in the driver queue according to the priority of the requesting task, or gives the packet to the driver (if bit UC.QUE is set--see Section 4.1.4.1).

2.4.2 Post-Driver Initiation Services

Once a driver is given control following an I/O interrupt or by the Executive itself, a number of Executive services are available to the driver. These services are discussed in detail in Chapter 5.

However, four Executive services merit special emphasis because virtually every driver in the system uses them:

- Interrupt Save ($INTSV)
- Get Packet ($GTPKT)
- Create Fork Process ($FORK)
- I/O Done ($IODON or $IOALT)

2.4.2.1 **Interrupt Save ($INTSV)**[1] - Interrupts from a device are fielded by the driver. Immediately following the interrupt, the driver operates at hardware priority level 7 and is, therefore, noninterruptable. If the driver needs a lengthy processing cycle (greater than 100 microseconds) to process the interrupt, or if it requires the use of any general-purpose registers, it calls $INTSV. This call queues external interrupts, alters the hardware priority, and provides the calling routine with two free registers to use in processing the interrupt. $INTSV is discussed in more detail in Section 2.5.

2.4.2.2 **Get Packet ($GTPKT)** - The Executive, after it has queued an I/O packet, calls the appropriate driver at its I/O-initiator entry point. The driver then immediately calls the Executive routine $GTPKT to obtain work.[2] If work is available, $GTPKT delivers to the driver the highest-priority, executable I/O packet in the driver's I/O queue, and sets the SCB status to "busy." If the driver's I/O queue is empty, $GTPKT returns a no-work indication. If the SCB related to the device is already busy, $GTPKT so informs the driver, and the driver immediately returns control to the Executive.

Note that, from the driver's point of view, no distinction exists between no-work and SCB busy, because an I/O operation cannot be initiated in either case.

2.4.2.3 **Create Fork Process ($FORK)** - You can synchronize access to shared data bases by creating a fork process. When a driver needs to access a shared data base, it must do so as a fork process; the driver becomes a fork process by calling $FORK. The SCB contains preallocated storage for a 4- or 5-word "fork block." See Section 4.1.3.1 for a description of the fork block. Section 5.3 contains details on $FORK.

---

1. A loadable driver on a mapped system cannot call $INTSV directly. See Section 4.3.

2. An exception is a driver that handles special user buffers. Such a driver must call certain other Executive routines before calling $GTPKT. See Section 6.3 for an example.

An interrupt routine cannot call $FORK directly; the routine must first switch to system state by using the INTSV$ macro or calling $INTSV (as described in Section 2.4.2.1). Furthermore, the interrupting routine's priority is lowered to that of the requesting source.

After calling $FORK, a routine is fully interruptable (priority 0), and its access to shared system data bases is strictly linear.

2.4.2.4 **I/O Done ($IODON or $IOALT)** - At the completion of an I/O request, the subroutines $IODON or $IOALT perform a number of centralized checks and additional functions:

- Store status if an IOSB address was specified
- Set an event flag, if one was requested
- Determine if a checkpoint request can now be honored
- Determine if an AST should be queued

$IODON and $IOALT also declare a significant event, reset the SCB and device unit status to "idle," and release the dynamic storage used by the completed I/O operation.

2.5 PROGRAMMING STANDARDS

RSX-11M I/O drivers function as integral components of the RSX-11M Executive. They must follow the same conventions and protocol as the Executive itself if they are to avoid complete disruption of system service. This manual has been written to enable you to incorporate I/O drivers into your system. Failing to observe the internal conventions and protocol can result in poor service and a reduction in system efficiency.

2.5.1 Process-Like Characteristics of a Driver

A driver is an asynchronous Executive process. As a process, it possesses its own context, allows or disallows interrupts, and synchronizes functions within itself (all drivers can be parallel, multiunit, multicontroller) and with other Executive processes executing in parallel.

Most RSX-11M drivers are small. Their small size is made possible by a comprehensive complement of centralized services available by calls to the Executive, and by the availability of an information-dense, highly formalized I/O data structure.

2.5.2 Programming Conventions

Appendix E of the PDP-11 MACRO-11 Language Reference Manual describes program coding standards. DIGITAL recommends that users refer to these standards to assist in preparing standards for their own installations.

2.5.3 Programming Protocol

Because an I/O driver accepts interrupts directly from the devices it controls, the central Executive relies on the driver to follow strict programming protocol so that system performance is not degraded. Furthermore, the protocol ensures that the driver properly distributes shared resources according to user-specified priorities. The protocol is summarized in Section 2.5.3.4.

When a device interrupts, an I/O driver is entered. The driver usually calls $INTSV or issues the INTSV$ macro[1] for common save and state-switching services. At the completion of the services provided by INTSV$ or $INTSV, the interrupt routine is again given control to complete the interrupt service. The exit routine $INTXT restores the state prior to switching to the system state, controls the unnesting of interrupts, and makes checks on the state of the system (for example, it checks to determine whether it is necessary to switch stacks). The fork processor causes access to shared system data bases to be linear. The details of all these routines are given in Chapter 5.

The interrupt vectors for each controller type in low memory contain a program counter (PC) unique to each interrupting source and a Processor Status Word (PSW) set with a priority of 7. It is a system software convention to use the low-order four bits of the PSW to encode the controller number for multicontroller drivers. When a device interrupt occurs, the hardware pushes the current PSW and PC onto the current stack and loads the new PC and PSW (set at priority 7 with the controller number in the condition-code bits) from the appropriate interrupt vector. The driver then starts executing with interrupts locked out. A driver itself may be executing at one of three levels of interrupt sensitivity:

1. At priority 7 with interrupts locked out.

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

3. At fork level, which is at priority 0.

2.5.3.1 **Processing at Priority 7 with Interrupts Locked Out** - By internal convention, processing at this level is limited to 100 microseconds. If processing can be completed in this time, either without using general-purpose registers or by saving and restoring the registers used, then the driver simply dismisses the interrupt by executing an RTI instruction. The interrupt is processed and dismissed with minimal overhead.

2.5.3.2 **Processing at the Priority of the Interrupting Source** - If the driver requires additional processing time or requires the use of general-purpose registers, it calls the $INTSV routine. Loadable drivers on mapped systems must use the INTSV$ macro. All other drivers can use the INTSV$ macro or call the $INTSV routine directly. The priority at which the caller is to run is included in the call to the $INTSV routine. The driver sets this priority level to that of the interrupting source.

---

1. The system macro INTSV$ simplifies the coding of standard interrupt-entry processing (see Section 4.3).

The $INTSV routine sets up the interrupt routine so that it can be interrupted by devices with priorities higher than that of the interrupting source, and switches to system state if the processor is not already in system state.

The $INTSV routine also saves registers R4 and R5 to free these registers for the driver. (A standard practice is for the driver to set R4 to the address of the interrupting device's SCB, and R5 to its UCB address.) An internal programming convention assumes that the driver will not require more than these two registers during interrupt processing. If it does, the driver must save and restore any additional registers it uses. Processing time following the return from the $INTSV routine should not exceed 500 microseconds.[1]

NOTE

> In actual practice, every driver in the system calls the $INTSV routine on every interrupt, due to three factors:
>
> 1. It is difficult to service an interrupt without using one or two registers.
>
> 2. Most interrupt code can safely be executed at the priority of the interrupting source. Executing at that priority is more desirable in terms of system response than continuing to execute at the highest priority.
>
> 3. The $INTSV routine is an integral part of the interrupt mechanism for loadable drivers.

2.5.3.3 **Processing at Fork Level** - A driver calls $FORK to become fully interruptable (so that it conforms to the 500-microsecond time limit), or to access the shared system data base. The INTSV$ macro must be issued or the $INTSV routine must be called prior to calling $FORK.

By calling $FORK, the routine becomes fully interruptable and its access to system data bases is strictly linear. At fork level, all registers are available to the process, and R4 and R5 retain the contents they had on entrance to $FORK.

---

1. The 500-microsecond period is a guideline. The longer the period of time a real-time executive spends at an elevated priority level, the less responsive is the system to devices of equal or lower priority. This guideline is especially important if the device being serviced is at the same or higher priority than a character-interrupt device such as the DU11, which is vulnerable to data loss due to interrupt lockout.

2.5.3.4 **Programming Protocol Summary** - Drivers are required to adhere to the following internal conventions when processing device interrupts:

1. No registers are available for use unless $INTSV has been called or the driver explicitly performs save and restore operations. If $INTSV is called, registers R4 and R5 are available; any other registers must be saved and restored.

2. Noninterruptable processing must not exceed twenty instructions, and processing at the priority of the interrupting source must not exceed 500 microseconds.

3. You must use a fork process for all modifications to system data bases.

2.6 FLOW OF AN I/O REQUEST

Following an I/O request through the system at the functional level (the level at which this chapter is directed) requires that limiting assumptions be made about the state of the system when a task issues a QIO directive. The following assumptions apply:

- The system is up and ready to accept an I/O request. All required data structures for supporting devices attached to the system are intact.

- The only I/O request in the system is the sample request under discussion.

- The example progresses without encountering any errors that would prematurely terminate its data transfer; thus, no error paths are discussed.

The I/O flow proceeds as follows:

1. [Task issues QIO directive]

   All Executive directives are called by means of EMT 377. The EMT causes the processor to push the PSW and PC on the stack and to pass control to the Executive's directive processor.

   a. First-level validity checks

      The QIO directive processor validates the LUN and UCB pointer.

   b. Redirect algorithm

      Because the UCB may have been dynamically redirected by an MCR Redirect command, QIO directive processing traces the redirect linkage until the target UCB is found.

   c. Additional validity checks

      The EFN and the address of the I/O status block (IOSB) are validated. The event flag is reset and the I/O status block is cleared.

2. [Executive obtains storage for and creates an I/O packet]

   The QIO directive processor now acquires an 18-word block of dynamic storage for use as an I/O packet. It inserts into the packet data items that are used subsequently by both the Executive and the driver in fulfilling the I/O request. Most items originate in the requesting task's Directive Parameter Block (DPB).

3. [Executive validates the function requested]

   The function is one of four possible types:

   - Control
   - No-op
   - ACP
   - Transfer

   Control functions are queued to the driver. If the function is IO.KIL, the driver is called at its cancel I/O entry point. The IO.KIL request is then completed successfully.

   No-op functions do not result in data transfers. The Executive "performs" them without calling the driver. No-ops return a status of IS.SUC in the I/O status block.

   ACP functions may require processing by the file system. More typically, the request is a read or write virtual function that is transformed into a read or write logical function without requiring file system intervention. When transformed into a read or write logical, the function becomes a transfer function (by definition). See Appendix D for more information on ACP functions.

   Transfer functions are address checked and queued to the proper driver. Then the driver is called at its initiator entry point.

4. [Driver processing]

   a. Request work

      To obtain work, the driver calls the $GTPKT routine. $GTPKT either provides work, if it exists, or informs the driver that no work is available, or that the SCB is busy. If no work exists, the driver returns to its caller. If work is available, $GTPKT sets the device controller and unit to "busy," dequeues an I/O request packet, and returns to the driver. If the I/O request is IO.ATT or IO.DET, the request is processed by the Executive without returning the packet to the driver, unless UC.QUE is set.

      If UC.QUE is set, the packet is passed to the driver after the IO.ATT or IO.DET processing is completed. If the request is to be processed by an ACP, the packet is queued to the ACP. In either case, $GTPKT will look for another request for the driver.

b. Issue I/O

From the available data structures, the driver initiates the required I/O operation and returns to its caller. A subsequent interrupt may inform the driver that the initiated function is complete, assuming the device is interrupt driven.

5. [Interrupt processing]

When a previously issued I/O operation interrupts, the interrupt causes a direct entry into the driver, which processes the interrupt according to the programming protocol described in Section 2.5. According to the protocol, the driver may process the interrupt at priority 7, at the priority of the interrupting device, or at fork level. If the processing of the I/O request associated with the interrupt is still incomplete, the driver initiates further I/O on the device (step 4b). When the processing of an I/O request is complete, the driver calls $IODON.

6. [I/O Done processing]

$IODON removes the "busy" status from the device unit and controller, queues an AST, if required, and determines if a checkpoint request pending for the issuing task can now be effected. The IOSB and event flag, if specified, are updated, and $IODON returns to the driver. The driver branches to its initiator entry point and looks for more work (step 4a). This procedure is followed until the driver finds the queue empty, whereupon the driver returns to its caller.

Eventually, the processor is granted to another ready-to-run task that issues a QIO directive, starting the I/O flow anew.

2.7 DATA STRUCTURES AND THEIR INTERRELATIONSHIPS

This section introduces all the individual control blocks, as well as their linkages and use within the system. The following data structures comprise the complete set for I/O processing:

1. Task header
2. Window Block (WB)
3. File Control Block (FCB)
4. $DEVHD word, the Device Control Block (DCB), and the Driver Dispatch Table (DDT)
5. Unit Control Block (UCB)
6. Status Control Block (SCB)
7. Volume Control Block (VCB)

Figure 2-5, which provides the structure for the following discussion, shows all the individual data structures and the important link fields within them. The numbers on the figure are keyed to the text to simplify the discussion and to guide the reader through the data structures.

1. The task header is constructed during the task-build process.[1] (It is one of two independent entries in the I/O data structure, the other being $DEVHD.) The task header entry of interest, the Logical Unit Table (LUT), is allocated by the Task Builder and filled in at task installation. The number of LUT entries is established by the UNITS= keyword option; this number is an upper limit on the number of device units a task may have active simultaneously. Each LUT entry contains a pointer to an associated UCB, and a pointer to a Window Block if a file is accessed by that logical unit number (LUN).

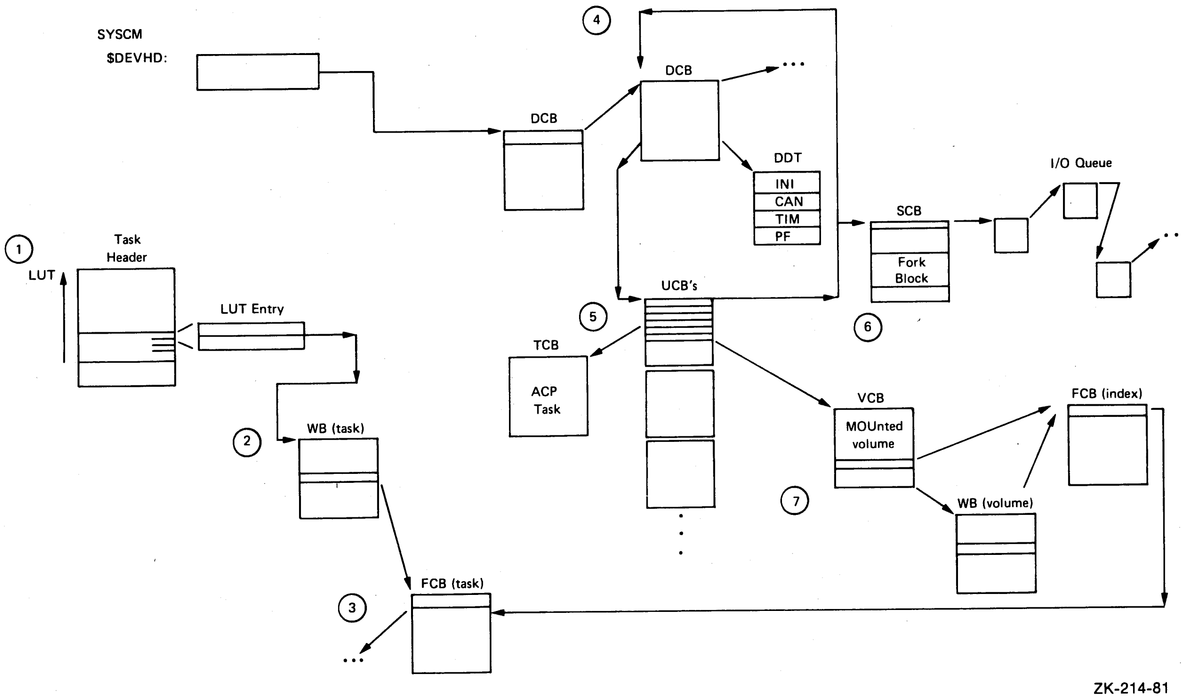


Figure 2-5 I/O Data Structure

2. A Window Block (WB) exists for each active access to files on a mounted volume. It contains the context for the virtual patch used for validating I/O requests and converting virtual functions to logical functions. For disks, the WB consists of pointers to contiguous areas on the device.

3. An FCP is a data structure specific to the Files-11 disk ACP (F11ACP). It is used to control access to the file.

---

1. In mapped systems, a copy of the task header (located in the task's partition) is made in the Executive's dynamic storage region. The Executive then uses this copy. To access the current information in this copy, a task must be privileged and have mapping to the Executive.

4. $DEVHD is a word located in system common (SYSCM) and is the other independent entry in the I/O data structure. $DEVHD points to a singly linked, unidirectional list of Device Control Blocks (DCBs). Each device type in a system has at least one associated DCB. The DCB list is terminated by a zero in the link word.

   Linked to each DCB is a Driver Dispatch Table (DDT), which is part of the driver. The DDT contains the addresses of the driver's four entry points that the Executive can call.

5. A key data structure is the Unit Control Block (UCB). All the UCBs associated with a DCB appear in consecutive memory locations. During internal processing of an I/O request, most drivers set R5 to the address of the related UCB. It is by means of pointers in the UCB that other control blocks in the data structure are accessed. In particular, the UCB contains pointers to the DCB, SCB, VCB, and to the UCB to which it may have been redirected. If a Redirect command has not been issued for the device-unit, the UCB redirect pointer points to the UCB itself. When servicing a request for one of its UCBs, the driver is unaware of whether I/O was issued directly to its own UCB or to a UCB that had been redirected to its UCB.

6. One Status Control Block (SCB) exists for each controller in a system. A unique SCB must exist for each controller/device-unit capable of performing parallel I/O. The SCB contains the fork-block storage required when a driver calls $FORK to transfer itself to the fork-processing level. The I/O request queue listhead is also contained in the SCB. Generally, register R4 contains the address of the SCB during processing of an I/O request.

7. One Volume Control Block (VCB) exists for each mounted volume in a system. The VCB maintains volume-dependent control information.

   For Files-11 disks, the VCB contains pointers to the File Control Definition Block (FCB) and Window Block (WB), which control access to the volume's index file. (The index file is a file of file headers.) The WB for the index file serves the same function as the WB for a user file. (See the <u>IAS/RSX-11 I/O Operations Reference Manual</u> for more information on the index file.) All unique FCBs for a volume form a linked list emanating from the index file FCB. This linkage aids in keeping file access time short. Further, since the window that contains the mapping pointers is variable in length, the user can increase file access speed by adding more access pointers (greater speed requires more main memory) to whatever extent the application requires.

2.7.1 Data Structures Summary

As the writer of a conventional driver, you do not manipulate the entire I/O data structure. In fact, you are usually involved only with the I/O packet, the UCB, and the SCB. The entire structure has been presented to add depth to your understanding of the I/O system, to emphasize the relationships among individual control blocks, and to clarify further the role a given driver fulfills in the processing of an I/O request.

CHAPTER 3

INCORPORATING USER-WRITTEN DRIVERS INTO RSX-11M

If you want to support an I/O device for which DIGITAL has not supplied a driver, you can write your own driver. Although this manual has not yet presented explicit details for writing such a driver, you now have enough conceptual information to consider incorporating one of your own drivers into your system. As will be seen, many considerations for writing a driver are best presented in a discussion of incorporating one.

NOTE

> An alternative approach to writing your own device driver may be the CINT$ (Connect to Interrupt Vector) directive. Refer to the description of the CINT$ directive in the RSX-11M/M-PLUS Executive Reference Manual. For examples of the use of CINT$, examine the task-level support routines for K-series laboratory peripheral modules, as described in Chapter 22 of the RSX-11M/M-PLUS I/O Drivers Reference Manual.

3.1 OVERVIEW OF INCORPORATING USER-WRITTEN DRIVERS

How you incorporate a user-written driver into the system depends mainly on whether you make your driver loadable or resident. If your driver is loadable, its data base can be either loadable or resident. If your driver is resident, both its data base and its code are resident. Thus, because you build the Executive image during system generation, you must include all resident driver elements in the Executive image during system generation. If your driver is loadable and has a loadable data base, you can incorporate it at any time after you build the Executive under which the driver runs.

3.1.1 System Generation Support for User-Written Drivers

During system generation, you answer questions concerning the types and quantity of peripheral devices on your system. Based on your answers, SYSGEN creates a single source file SYSTB.MAC that constitutes the data base definitions for DIGITAL-supplied devices on the system. Also created are the object modules of driver code to support the devices. Assembling SYSTB.MAC creates one module, SYSTB.OBJ, which becomes the system device tables for the Executive and the data base for the system drivers. This module is linked into and becomes a permanent part of the Executive.

A privileged system task invoked with the MCR/VMR LOA command is responsible for loading into memory a driver that is not resident. The LOA function creates the necessary interrupt control blocks (ICB) for accessing a driver in a mapped system, and establishes the linkage between the data base structures in the system device tables and the driver code being loaded. Another privileged system task invoked with the MCR/VMR UNL command can remove a loadable driver from memory. (Although the UNL function removes a loadable driver, it does not remove a loadable data base.)

SYSGEN asks questions concerning the necessary driver support features in the Executive, to allow you to add user-written drivers.

During the Target Configuration section of SYSGEN Phase I, answer the following question with the highest vector your system will use with the addition of your user-written driver:

14. Highest interrupt vector [O R:0-774 D:0]:

SYSGEN uses the specified address or 400, whichever is greater, to allocate sufficient vector space in the Executive to avoid run-time destruction of the system stack and to avoid hardware trapping (which occurs when the SP goes below 400).

If any user-written driver is to be built loadable, SYSGEN must be notified of this fact. Loadable drivers require extra Executive features to support them (for example, the MCR/VMR LOA and UNL commands). You can choose support for loadable drivers by answering the following question in the Executive Options section of SYSGEN Phase I:

15. Loadable device drivers? [Y/N]:

The answer to the following question determines whether SYSGEN allows you to include a user-written driver in the system:

25. Do you intend to include a user-written driver? [Y/N]:

If you answer Yes, the next two questions are also asked (see Section 5.3 of this manual):

26. Include routine $GTWRD? [Y/N]:
27. Include routine $PTWRD? [Y/N]:

NOTE

> If an LPA11 device (LA:) is included in your system, the $GTWRD routine is automatically included and Question 26 is not asked. If an AD01 A/D controller device or an AFC11 A/D controller device (AF:) is included in your system, the $PTWRD routine is automatically included and Question 27 is not asked. The $GTWRD and $PTWRD routines are described in Chapter 5 of this manual.

To incorporate a user-written driver into RSX-11M, you first create two modules, one in which you define the data base and the other in which you include the driver code itself. You then must link your driver data base and driver code modules into the system device tables in the Executive. The linkage that your data base module must satisfy is the link of the Device Control Block (DCB) list. The linkage for the driver code connects the DCB for the device that your driver supports to the Driver Dispatch Table (DDT). Moreover, if your driver is loadable, you must supply in your driver code symbols and labels that the LOA command needs.

If your driver is loadable and has a loadable data base, you build (1) a loadable image containing the driver code module followed by the driver data base module, and (2) a symbol definition file on which the LOA command depends to find critical data base and driver locations. You build the driver image with the symbol definition file of the Executive under which the driver runs. However, the driver image is separate from the Executive image. The LOA command is responsible for loading both your driver data base and driver code, for connecting the data base to the system device tables, and for connecting your driver code to the data base.

If your driver is loadable but has a resident data base, you must perform SYSGEN and build the Executive under which the driver runs to link your driver data base module(s) into the system device tables. This operation makes your driver data base resident with the system device tables. You also build (1) a loadable image containing the driver code, and (2) a symbol definition file which the LOA command uses to locate the driver dispatch table. The LOA command is responsible for loading your driver code and for connecting your driver code to the data base that is resident with the system device tables.

If your driver is resident, you must perform a system generation and build the Executive to link the driver data base into the system device tables and to include the driver code in the Executive image.

Because the LOA command provides consistency and validity checks on a data base being loaded, DIGITAL recommends that you make your driver and its data base loadable. Furthermore, with a loadable driver and loadable data base, you can more easily modify your driver and its data base. You need not rebuild your Executive and privileged tasks. To change the driver code, you need only build a new driver image, unload the current version, and reload the new version. To change the driver data base, you must build a new driver image (which incorporates the data base module), rebootstrap your system, and load the new driver to load the modified data base. (You must bootstrap your system to change the data base because the UNL command does not unload a data base, and because the LOA command does not load a data base for a driver if one is currently loaded for that driver.)

NOTE

> If you use VMR to load the data base into the system image, rebooting the system will always load the data base.

Using a loadable driver with a loadable data base may make more work in the short term but saves work in the long term. During debugging, data base inconsistencies are likely to be caught by the LOAD command. Thus, you prevent many such errors from later creating system problems. The procedure to make your driver operational is more complex because both the driver (and its data base) must be loaded each time the system starts up.

A resident driver or a loadable driver with a resident data base is easier to incorporate into the system but more difficult to debug and to modify. The LOA command does not perform consistency and validity checks on a resident data base. Thus, a debugging aid is not available. Moreover, to modify such drivers, you must rebuild the Executive, which generally implies rebuilding the privileged tasks.

The capability to incorporate a user-written driver into your system is a supported feature of RSX-11M. Because a user-written driver is considered a system modification, DIGITAL may not support the system that results after you incorporate your driver. Being a part of the Executive, your driver can subtly corrupt it. Therefore, DIGITAL cannot guarantee support which entails debugging user-written drivers.

Fixing a problem in a system is largely a matter of being able to reproduce the problem reliably. If a problem on your system can be shown to have no relation to your driver, and DIGITAL will reproduce the problem, SPR support can be provided. A good reason for using a loadable driver with a loadable data base is that you can more easily test an unmodified system by not loading your driver and its data base. You can then reproduce a suspected problem in an unmodified system and can submit an SPR that DIGITAL will answer. Therefore, attempting to re-create a suspected problem on your system without your driver and its data base saves both you and DIGITAL time in answering the SPR.

3.1.2 Overview of User-Written Driver Code

To create the source code to drive a device, you must do the following:

1. Thoroughly read and understand this manual.

2. Familiarize yourself in detail with the physical device and its operational characteristics.

3. Determine the level of support required for the device.

4. Create the data base source code for the device.

5. Determine actions to be taken at the five driver entry points:

   - Initiator
   - Cancel I/O
   - Time-out
   - Powerfail
   - Interrupt

6. Create the driver source code. This code will contain the following global symbols (where xx is the 2-character alphabetic device mnemonic):

$xxTBL::	Address of the driver dispatch table (see Section 4.1.2.1)
$xxINT::	Address of the driver interrupt entry point
$xxINP:: $xxOUT::	Addresses of input and output interrupt entry points (for full-duplex devices)

   If a loadable driver's loadable data base is to include the interrupt vector(s) for the case when the driver is resident, the conditional assembly symbol LD$xx must be included in the source code for the loadable data base. In addition, the code in the data base which includes the interrupt vector(s) must be enclosed in a conditional assembly block and defined as an ASECT.

   Loadable drivers have an additional requirement. Either within the driver source code itself or in file RSXMC.MAC, the conditional assembly symbol LD$xx must be defined. The INTSV$ macro (see Section 4.3) uses this symbol (and others in RSXMC.MAC) to determine whether the call to $INTSV should be omitted from the driver.

   The symbols used to name interrupt entries are different for devices that employ error logging. See the RSX-11M/M-PLUS Error Logging Reference Manual for information on modifying device drivers for error logging. Note that the error logger must be modified to log errors for a device that uses a user-written driver.

   The DIGITAL-supplied terminal driver (TTDRV) is treated as a special case by the LOA command in terms of the naming of its interrupt entries.

3.1.3 Overview of User-Written Driver Data Bases

Of the data structures associated with an I/O driver, four require assembly source code:

- The DCB
- The UCB(s)
- The SCB(s)
- The device interrupt vector (assembly source required for resident drivers only)

A single DCB can service multiple UCBs and SCBs. The existence of multiple UCBs and SCBs is determined by the actual device subsystem being supported by a given driver in your hardware configuration. Figures 2-2, 2-3, and 2-4 illustrate possible DCB, UCB, and SCB structural relationships.

Within the DCB, UCBs, and SCBs, only those fields that are static or that need initialization must be supplied in your assembly source. Tables 3-1, 3-2, and 3-3 list these required fields. See Chapter 4 for detailed figures and descriptions of these and other data structures.

Table 3-1
Required DCB Fields

Offset	Description
D.LNK	Link to next DCB. This field is zero if this is the last (or only) DCB. If you are incorporating more than one user-written driver at one time, then this field should point to another DCB in a DCB chain.
D.UCB	Address of the first word (U.DCB) of the first UCB associated with this DCB.
D.NAM	Two-character ASCII generic device name.
D.UNIT	Highest and lowest logical unit numbers controlled by this DCB.
D.UCBL	Length of the UCB (including prefix words, if any). If a given DCB has multiple UCBs, all UCBs must be of the same length.
D.DSP	Address of the driver dispatch table. The dispatch table is located within the driver code. This field contains a global reference to the label associated with this table. The field is zero if the driver is loadable.
D.MSK	I/O function masks. You must supply bit-by-bit mapping for these four I/O function masks. Note that the format of the mask words is split into two groups of four words. Functions 0-15. are covered by the first group, and functions 16.-31. by the second.
D.PCB	Address of driver Partition Control Block (PCB). This field is required only if loadable driver support is included in the system. It must be initialized to zero.

Table 3-2
Required UCB Fields

Offset	Description
U.DCB	Backpointer to the associated DCB
U.RED	Redirect pointer--initially contains the address of this UCB
U.CTL	Control bits that must be established by the driver writer with the UCB source
U.STS	Unit status byte
U.UNIT	Physical unit number serviced by this UCB
U.ST2	Unit status byte extension
U.CW1	Characteristics word 1; standard description (see Section 4.1.4.1) applies
U.CW2	Driver-dependent
U.CW3	Driver-dependent
U.CW4	Default buffer size
U.SCB	Address of the SCB for this UCB
U.ATT	TCB address of attached task

Table 3-3
Required SCB Fields

Offset	Description
S.LHD	I/O queue listhead
S.PRI	Priority of interrupting source
S.VCT	Interrupt vector address divided by 4
S.ITM	Initial timeout count
S.CON	Controller index (that is, controller number multiplied by 2)
S.STS	Controller status
S.CSR	Address of control and status register
S.FRK	Fork block
S.MPR	Mapping register block; needed only by UNIBUS NPR devices running on a PDP-11 processor that employs extended-addressing (22-bit) mode

3.2 USER-WRITTEN LOADABLE DRIVERS

In deciding whether the data base for your loadable driver should be resident or loadable, consider the following characteristics of loadable data bases:

- When you load a driver, the MCR/VMR LOA command checks to see whether a data base is resident for the type of device whose driver is being loaded. If a data base is not resident, the LOA command reads the driver symbol definition file to find the start and end of the data base in the driver image. (Thus, if your driver data base is to be loadable, you must have defined its start and end in the data base source code.) Knowing the start and end, the LOA command reads the data base from the driver image. The LOA command places the data base in the system pool so that it resides in Executive address space, accordingly relocates pointers and links within the data base to be valid Executive addresses, and also connects DCB(s) in the data base to the system device tables. Moreover, the LOA command performs many consistency and validity checks on the data base being loaded so as to prevent the system device tables from being corrupted by an incorrect data base.

- A loadable data base is only loaded once; thereafter, it is resident until the system is bootstrapped again. The UNL command does not remove a data base from memory even though the data base was loaded with the LOA command.

- The LOA command relocates certain known pointers within the control blocks.[1] If the data base requires relocation of additional address pointers beyond the standard ones, it cannot be loaded with the LOA command. It must be incorporated into the system as a resident data base during system generation.

During debugging of a loadable driver (with loadable data base), you can correct errors in the coding of the driver itself by unloading, modifying, assembling, task-building, and reloading the driver. However, if the data base must be replaced, the system must be bootstrapped to remove it. You can then modify, assemble, and task-build the data base, and reload it along with the (corrected) driver.

The subsections below describe the procedure for incorporating a user-written loadable driver.

3.2.1 Creating the Loadable Data Base and Driver Modules

The general procedure for incorporating a loadable driver with a loadable data base is as follows:

1. Complete SYSGEN Phase I and answer the appropriate questions that include the necessary driver support features in the Executive. Edit the assembly prefix file RSXMC.MAC and define a conditional symbol LD$xx for each loadable driver. See Section 3.1.1 for a discussion of system generation support.

---

1. The pointers are: (DCB) D.LNK, D.UCB; (UCB) U.DCB, U.RED, U.SCB. (SCB) S.LHD+2. Chapter 4 gives details on these and other fields in the data base.

2. Complete SYSGEN Phase II. During Phase II you must edit RSXBLD.CMD, the Executive build command file (see Section 3.3), and add the following line:

   ```
   GBLDEF=$USRTB:0
   ```

   (The symbol $USRTB in the file [11,10]SYSTB.MAC defines the link word to a user-written driver's resident DCB. Adding this line forces the link word to be zero.) If you do not add this line when you do not have a resident data base, the Task Builder generates an undefined symbol error when it builds the Executive. However, you can disregard that error because the Task Builder sets the contents of the link word to zero.

3. After SYSGEN Phase II completes, you can manually build the driver.

After completing these steps, you can assemble the driver and data base at any time.

4. While both the loadable driver and its data base can be contained in the same source module, it is recommended that you create separate sources for your driver and its data base and place them in UFD [11,10]. If, however, you place both the data base and the driver in the same module, you must ensure that, when linked, the data base follows the driver code. You can do this by physically placing the data base code after the driver code or by using .PSECT names to force proper allocation.

5. A useful convention is to name your data base source xxTAB and your driver source xxDRV, where xx is the 2-character (alphabetic) device mnemonic.

6. You must place the DCB first in the assembly source code of the driver data base module. In a multiuser protection system, the DCB must be followed first by the associated UCB(s) and then by the SCB(s). All UCB(s) associated with a particular DCB must be contiguous. DIGITAL-supplied drivers use this ordering scheme; see the file [11,10]SYSTB.MAC, created by Phase I of SYSGEN, for examples. Since you are creating a loadable data base for a single driver, your source code will contain a single DCB with associated UCB(s) and SCB(s).

7. The global label $xxDAT:: marks the start of your driver's data base (the DCB). The global label $xxEND:: marks the end of the data base (that is, immediately following the final word of the data base). These labels are absolutely required. The letters xx represent the 2-character device mnemonic.

To assemble your driver, set the default UIC to [11,2x] and run the assembler as follows (user input underlined):

```
>SET /UIC=[11,2x] (RET)
>MAC (RET)
MAC>xxDRV,xxDRV=LB:[1,1]EXEMC/ML,LB:[11,10]RSXMC,xxDRV(RET)
```

To assemble the data base, use the following input to MAC:

```
MAC>xxTAB,xxTAB=LB:[1,1]EXEMC/ML,LB:[11,10]RSXMC1,xxTAB (RET)
```

Next, you use the following command to add both the driver and its data base to the Executive object module library:

```
LBR>LB:[1,2x]RSX11M/RP=[11,2x]xxDRV,xxTAB (RET)
```

3.2.2 Task-Building a Loadable Driver

In this section, two examples of task-building a loadable driver with a loadable data base are presented: one for a mapped system and one for an unmapped system.

3.2.2.1 Task-Building a Loadable Driver on a Mapped System

The following seven requirements apply to task-building any loadable driver, whether user-written or DIGITAL-supplied.

1. You must specify a task image file name and a symbol definition file name as TKB output. Both files must be placed in the UFD corresponding to the system UIC that will be in effect when the command is issued. The file names must both be xxDRV, where xx is the device mnemonic. The Task Builder produces output files named xxDRV.TSK and xxDRV.STB. For example, the beginning of input to TKB to build a paper-tape reader driver for a mapped system might appear as follows (user input underlined):

```
>TKB (RET)
TKB>[1,54]PRDRV/-HD/-MM,,[1,54]PRDRV= (RET)

          ↑         ↑   ↑            ↑
     Task image. Switches: see    Symbol
                  items 2 & 3     definition.
                  below.
```

2. You must not have a task header. Use the switch /-HD, as in the example above. A driver is not really a task but an extension of the Executive, and as such needs no task header.

3. You must use the /-MM switch, whether in fact the driver is destined for a mapped or an unmapped system.

4. You must link to the system symbol definition file that contains definitions of Executive global symbols. Continuing the paper-tape reader driver example referred to above, further TKB input might look like this:

```
TKB>LB:[1,24]RSX11M/LB:PRDRV:PRTAB (RET)
TKB>[1,54]RSX11M.STB/SS (RET)
```

   The first line above specifies the library file (/LB) in which the input driver object module and the object file for the loadable data base can be found. The object module specification for the driver must always precede the specification for the data base in the TKB command line.

   You omit the data-base file specifier when task-building any DIGITAL-supplied driver or one of your own drivers if it has a resident data base. All DIGITAL-supplied drivers that are declared loadable at system generation use resident data bases.

The second line in item 4 above indicates that the symbol table file RSX11M.STB is to be searched selectively (/SS) for definitions of Executive global symbols. Note that the /SS switch must appear in this context. It cannot be omitted.

5. You must link to the system library file that defines masks and offsets used in the Executive. Continuing the example:

```
TKB>LB:[1,1]EXELIB/LB (RET)
TKB>/ (RET)
```

The single slash begins the option phase of the Task Builder.

6. The driver code will execute as a part of the Executive, and thus the driver will use the Executive stack. Therefore, you must direct the Task Builder not to allocate space for a stack within the driver, as follows:

```
TKB>STACK=0 (RET)
```

7. You must specify a partition for the driver. The specification differs for mapped and unmapped systems. Continuing the mapped-system example:

```
TKB>PAR=DRVPAR:120000:4000 (RET)
TKB>// (RET)
>
```

On mapped systems, the starting address of the partition must be 120000 octal. That is, the loadable driver must be mapped to kernel APR 5.

On unmapped systems, the second parameter must be the physical starting address of the partition.

On either mapped or unmapped systems, the length of the partition may not exceed 4K words (20000 octal bytes).

The double slash terminates the Task Builder's input.

3.2.2.2 **Task-Building a Loadable Driver on an Unmapped System** - In the example below, we build a magtape driver for an unmapped system. The only differences from the mapped-system example are the partition starting address and the UFD of some of the files ([1,50] and [1,20] instead of [1,54] and [1,24], respectively).

```
>TKB (RET)
TKB>[1,50]MTDRV/-HD/-MM,,[1,50]MTDRV= (RET)
TKB>LB:[1,20]RSX11M/LB:MTDRV:MTTAB (RET)
TKB>[1,50]RSX11M.STB/SS (RET)
TKB>LB:[1,1]EXELIB/LB (RET)
TKB>/ (RET)
ENTER OPTIONS: (RET)
TKB>STACK=0 (RET)
TKB>PAR=DRVPAR:34000:4000 (RET)
TKB>// (RET)
>
```

3.2.3 Loading a User-Written Loadable Driver

Loading is done by using the privileged MCR command LOA. Its form is:

```
LOA xx:[/PAR=partition]
```

where xx is the 2-character device mnemonic. Specifying a partition is optional. If none is specified, the partition input to the Task Builder is used.

The LOA command requires that the two files xxDRV.TSK and xxDRV.STB reside under the system UFD (that is, the UFD established by the SET /SYSUIC command). Typically, this UFD is [1,50] for unmapped systems and [1,54] for mapped systems.

LOA searches first for a resident data base for the driver being loaded. If none is found, LOA looks for the following global symbols in the symbol definition file xxDRV.STB:

$xxDAT:: Address of the start of the data base (the DCB) associated with the driver

$xxEND:: Address+2 of the last word of the data base associated with the driver

3.2.4 Creating the Loadable Driver and Resident Data Base Modules

If you decide upon a resident data base, you create the data base object module during SYSGEN Phase I. You use the same procedure as that for a resident driver.

To create the resident data base, follow the procedure described in Section 3.3 with the exception of initializing the device interrupt vector. Since there is only one resident data base module (USRTB, which contains all the resident data bases), you assemble the resident data bases for loadable drivers with the resident data bases for resident drivers.

To assemble the loadable driver, use the following command to MAC:

```
MAC>xxDRV,xxDRV=LB:[1,1]EXEMC/ML,LB:[11,10]RSXMC,xxDRV<RET>
```

3.2.5 Building a Loadable Driver and Its Resident Data Base

You build all resident data bases into the Executive as described in Section 3.3. You build the loadable drivers after the Executive is built. When SYSGEN asks the following question during Phase II:

5. Driver 2-character device mnemonic [S]:

enter the characters xx, where xx is the driver name.

After your system is built, you can load your driver as described in Section 3.2.3.

3.3 USER-WRITTEN RESIDENT DRIVERS

This section describes specific details for user-written resident drivers.

1. Create the assembly source file for the resident data base in UFD [11,10]. Use USRTB.MAC as the file specification. USRTB as the file name is not actually required. It is, however, a useful convention -- as reflected in the sample dialogue described below.

2. There is no mandatory ordering of the different control blocks in the data base for your resident driver. It is suggested that you follow the convention of placing the DCB first, followed by the UCB(s), followed by the SCB(s). However, it is required that all UCB(s) associated with a particular DCB must be contiguous. DIGITAL-supplied RSX-11M drivers use this ordering scheme; see the file [11,10] SYSTB.MAC, created by Phase I of SYSGEN, for examples. If you are incorporating multiple resident drivers into your system, you will have more than one instance of a DCB with UCB(s) and SCB(s).

3. Initialize the device interrupt vector (refer to Section 4.1.5 for a description of this process).

4. Use the global label $USRTB:: as the address of your first (or only) DCB. This is absolutely required.

During Phase I, SYSGEN asks:

    25. Do you intend to include a user-written driver? [Y/N]

If you answer Yes, then subsequent questions guide you through the process of adding the driver to the generated system. Refer to Section 3.1.1 for a discussion of system generation support and the questions asked. Operations performed include assembling the driver and its data structure, and editing the RSX-11M task-build command file.

The following sample dialogue illustrates the addition of a resident driver for a PK: device. All user responses are underlined. After SYSGEN assembles the DIGITAL-supplied drivers, it prints some instructions, then pauses to allow you to assemble your driver(s), as follows:

```
>; Assemble user-written driver(s)
>;
>;      The following instructions apply to resident drivers and
>;      loadable drivers with resident data bases.
>;
>;      For loadable drivers, you must ensure that a symbol definition
>;      of the format:
>;          LD$xx=0
>;      (where xx is the device name) appears in the assembly prefix
>;      file [11,10]RSXMC.MAC for each loadable driver xxDRV.
```

```
>;
>;      SYSGEN will now pause to allow you to assemble your drivers(s)
>;      and USRTB module. Using a driver name xxDRV (where xx is
>;      the device name; for example, DK), you can type commands
>;      in the following format to assemble the driver and USRTB
>;      modules.
>;
>;     MAC
>;     MAC>xxDRV=LB:[1,1]EXEMC/ML,SY:[11,10]RSXMC,xxDRV
>;     MAC>USRTB=LB:[1,1]EXEMC/ML,SY:[11,10]RSXMC,USRTB
>;     MAC>^Z

AT. -- PAUSING. TO CONTINUE TYPE "RES ...AT."

>
>MAC (RET)
MAC>PKDRV=LB:[1,1]EXEMC/ML,SY:[11,10]RSXMC,PKDRV (RET)
MAC>USRTB=LB:[1,1]EXEMC/ML,SY:[11,10]RSXMC,USRTB (RET)
MAC>(CTRL/Z)
>
>RES ...AT.(RET)

AT. -- CONTINUING
>
```

After you exit from MACRO-11 and type the command to resume, SYSGEN finishes Phase I.

When you start Phase II, SYSGEN asks a few questions, prints some instructions, and pauses for you to build the driver(s) as follows:

```
>;
>; Build user-written driver(s)
>;
>;      You must now edit the Executive build command file RSXBLD.CMD
>;      to include your user-written driver and data base in your system.
>;
>;      If you are including a resident data base, locate the line
>;      in which the module SYSTB is referenced and add :USRTB
>;      immediately after it, for example:
>;           LB:[1,24]RSX11M/LB:SYSTB:USRTB:SYTAB:INITL,LB:[1,1]EXELIB/LB/SS
>;      If you are not including a resident data base, add the line
>;           GBLDEF=$USRTB:0
>;      to the file instead.
>;
>;      For each resident driver, add a line of the form:
>;           LB:[1,24]RSX11M/LB:xxDRV
>;      where other drivers are referenced (where xx is the device name,
>;      for example DK).
>;      NOTE:  For each loadable driver, do not add a corresponding
>;      line to the build command file
>;
>;      SYSGEN will now pause to allow you to edit RSXBLD.CMD.
>;      After you exit from the editor and resume, SYSGEN builds the
>;      Executive and drivers.

AT. -- PAUSING. TO CONTINUE TYPE "RES ...AT."

>EDI RSXBLD.CMD(RET)
[00028 LINES READ IN]
[PAGE    1]
*L SYSTB(RET)
LB:[1,24]RSX11M/LB:SYSTB:SYTAB:INITL,LB:[1,1]EXELIB/LB/SS
*C /SYSTB:/SYSTB:USRTB:/(RET)
```

```
LB:[1,24]RSX11M/LB:SYSTB:USRTB:SYTAB:INITL,LB:[1,1]EXELIB/LB/SS
*L DYDRV(RET)
[*EOB*]
*T (RET)
*L DYDRV(RET)
LB:[1,24]RSX11M/LB:DYDRV
*I(RET)
LB:[1,24]RSX11M/LB:PKDRV

*EX(RET)
[EXIT]

>RES ...AT.(RET)

AT. -- CONTINUING
```

After you perform the indicated operations and type the command to continue with SYSGEN, the Executive is built and you are given a chance to build any loadable drivers as follows:

```
>; Build Loadable drivers
>;
>*  4. Build all selected loadable drivers into DRVPAR? [Y/N]:Y
>;
>; You can now build your user-written driver (if it is a loadable
>; driver). If you choose not to build it now, or it is not loadable
>; strike carriage return in response to the next question.
>;
>; When all drivers are built, strike carriage return
>;
>* 5. Driver 2-character device mnemonic [S]:
```

When Phase II completes, your resident driver is incorporated in the Executive and is ready to run.

3.4 DRIVER DEBUGGING

Because the protection checks provided for user programs are not available to system modules, driver errors are more difficult to isolate than user-program errors. But conventional drivers, because they are modular and short, probably can be easily debugged. This debugging process requires that you understand the following topics, each of which is discussed in a separate subsection:

- Debugging aids and tools
- Fault isolation
- Fault tracing
- Rebuilding and reincorporating a driver

3.4.1 Debugging Aids

Adding a user-written driver carries with it the risk of introducing obscure bugs into an RSX-11M system. Since the driver runs as part of the Executive, special debugging tools are both desirable and necessary. RSX-11M provides several such aids, which can be incorporated into your system during system generation:

- Executive stack and register dump
- XDT
- Panic dump
- Crash Dump Analyzer (CDA) support routine.

You need not select any of this software during system generation. If, however, you do require the facilities they offer, you can select from one to three of them for incorporation in your system (panic dump and the CDA support routine are mutually exclusive). The following subsections describe the features and use of each debugging aid.

3.4.1.1 **Executive Stack and Register Dump Routine** - At system generation, you can indicate that you want a dump of the Executive stack and the registers when a crash occurs. If you choose this option, the system will perform as follows:

1. A system error, or the XDT X command (described in the next section), or operator manipulation of the switch register following a CPU halt, will cause processing to resume at location 40(octal).

2. Location 40(octal) contains a JMP instruction that causes the Executive to execute the code beginning at location $CRASH.

3. $CRASH invokes the routine that dumps the Executive stack and registers as shown below:

```
SYSTEM CRASH AT LOCATION 047622

REGISTERS

 R0=000340 R1=177753 R2=000353 R3=000000

 R4=000004 R5=046712 SP=000472 PS=000340

SYSTEM STACK DUMP

 LOCATION CONTENTS

 000472    000004
 000474    000000
 000476    001514
 000500    000340
 000502    177753
 000504    000353
 000506    000000
 000510    000000
 000512    057750
```

```
000514    002504
000516    030011
000520    100340
000522    000340
000524    056446
000526    000000
000530    102144
000532    000000
000534    101376
000536    101372
000540    102022
000542    170000
```

Following this display, either the CDA support routine or panic dump is invoked, depending on which (if either) is present in the system. Otherwise, the system halts.

3.4.1.2 **XDT - The Executive Debugging Tool** - An interactive debugging tool has been developed for RSX-11M to aid in debugging Executive modules, I/O drivers, and interrupt service routines. This debugging aid, called XDT, is a version of the RSX-11 Octal Debugging Tool (ODT). XDT does not contain the following features and commands available on ODT:

No $M - (Mask) register

No $X - (Entry Flag) registers

No $V - (SST vector) registers

No $D - (I/O LUN) registers

No $E - (SST data) registers

No $W - (Directive Status Word) $DSW word

No E - (Effective Address Search) command

No F - (Fill Memory) command

No N - (Not word search) command

No V - (Restore SST vectors) command

No W - (Memory word search) command

In addition, the X (Exit) ODT command has been changed for XDT. The X command causes a jump to the crash-reporting routine.

Except for the omitted features and the change in the X command, XDT is command-compatible with RSX-11 ODT; consequently, the <u>IAS/RSX-11 ODT Reference Manual</u> can be used as a guide to XDT operation.

XDT may be included in a system during Phase I of SYSGEN. The following question is asked:

*30. Executive Debugging Tool (XDT)? [Y/N]:

If you answer Yes, then XDT is automatically included in the generated system. When the resultant system is bootstrapped, XDT takes control and types on the console terminal:

```
XDT: <system version>
```

followed by the prompting symbol

```
XDT>
```

You can set breakpoints at this time, and then type the G command, passing control to the RSX-11M Executive initialization code. Whenever control reaches a breakpoint, a printout similar to that produced by RSX-11 ODT occurs.

You can initiate a forced entry to XDT at any time from a privileged terminal by means of the MCR BRK (breakpoint) command. Thus, the normal procedure is to type G when the system is bootstrapped without setting any breakpoints. When it becomes necessary to use XDT, the MCR BRK command is executed, causing a forced breakpoint. XDT then prints on the console terminal:

```
BE:xxxxxx
```

followed by the prompting symbol

```
XDT>
```

You can then set breakpoints and/or issue other XDT commands. Continue system operation by typing the P (proceed) command to XDT.

Note that XDT runs entirely at priority level 7 and does not interfere with user-level RSX-11 ODT. In other words, user-level RSX-11 ODT can be in use with several tasks, while XDT is being used to debug Executive modules.

All XDT command I/O goes to and from the console terminal, and the L (List Memory) command can list on either the console or the line printer.

Using XDT to debug a loadable driver on a mapped system has special pitfalls. One problem that can arise is a T-bit error:

```
TE:xxxxxx
XDT>
```

This error results when control reaches a breakpoint that you have set, using XDT, in a loaded driver on a mapped system. The T-bit error, rather than the expected BE: error, occurs unless register APR5 is mapped to the driver at the time XDT sets the breakpoint.

To avoid this T-bit error, assemble the driver with an embedded BPT instruction, or use either the ZAP utility or the MCR OPE command to set the breakpoint by replacing a word of code with the BPT instruction. When control reaches a breakpoint set in this manner, XDT prints:

```
BE:xxxxxx
XDT>
```

Recover as follows: Using XDT, replace the BPT instruction with the desired instruction. Decrement the PC by subtracting 2 from the contents of register R7. Then proceed by using the P or S commands.

NOTE

You should not set breakpoints in more than one module that maps into the Executive through APR 5 or APR 6. In particular, do not set breakpoints in more than one loadable driver at a time or XDT will overwrite words of main memory when it attempts to restore the contents of breakpoints.

3.4.1.3 **Panic Dump** - The Panic Dump routine saves registers R0 through R6 and then halts, awaiting dump limits to be entered.

The procedure for entering dump limits depends on whether your processor has a console switch register. The subsections that follow describe the procedure in each instance.

3.4.1.4 **Using the Panic Dump Routine on Processors with Console Switch Registers** - For processors with console switch registers, you can obtain dumps of selected blocks of memory by using the following procedure:

1. Enter the low dump limit in the console switch register and press CONT. The processor will immediately halt again.

2. Enter the high dump limit in the console switch register and press CONT. The dump will begin on the device whose CSR address is D$$BUG (usually 177514, which is the line printer). The actual value of D$$BUG is determined during system generation when answering the panic dump question. At the end of the dump, the processor will again halt, awaiting the input of another set of dump limits.

   If your system does not have the Executive routine stack and register dump, enter the dump limits 0-520(octal) when the Panic Dump routine first halts. This causes dump of the system stack and the general registers. The limit 520(octal) changes if the highest interrupt vector is above 400(octal). The actual upper limit is always the value of the global symbol $STACK and can be obtained from the global symbol listing in the Executive memory allocation map.

3.4.1.5 **Using the Panic Dump Routine on Processors Without Console Switch Registers** - A number of PDP-11 processors are being delivered without a console switch register; they are configured with the M9301 Console Emulator and Bootstrap. This presents no problem for the normal operation of RSX-11M, because it does not require a switch register. However, the Panic Dump Routine usually reads its arguments from the switch register. In systems that have been generated for processors that have no switch register, the panic dump module has been altered to read its arguments from location 0. The following instructions are designed to allow you to enter the proper information to the Panic Dump Routine on processors equipped with the M9301 Console Emulator.

1. When the Panic Dump Routine halts, the RUN light will go out. At this time press and release the BOOT switch.

   The Console Emulator should display:

   ```
   XXXXXX XXXXXX XXXXXX nnnnnn
   $
   ```

   where nnnnnn is the address+2 of the HALT instruction.

2. You should enter the following:

   ```
   $L 0(RET)
   $D low-address(RET)
   $L nnnnnn(RET)
   $S (RET)
   ```

3. The processor should again halt. Press and release the BOOT switch.

   The Console Emulator should display:

   ```
   XXXXXX XXXXXX XXXXXX mmmmmm
   $
   ```

4. You should then enter:

   ```
   $L 0(RET)
   $D high-address(RET)
   $L mmmmmm(RET)
   $S (RET)
   ```

At this time the dump should commence. When it is finished, the processor will halt and wait for additional input.

3.4.1.6 **Sample Output from Panic Dump** - A portion of the output from Panic Dump is shown below. Output is in 3-line groupings. In the left-hand column, two addresses are shown. The first address is the absolute address; the second address is the dump relative address. The first line in a 3-line group gives the octal word value; the second line gives the two octal byte values of the word; the third line contains the ASCII representation of the bytes. The ASCII representations in each word are reversed to improve readability. The first output grouping from Panic Dump shows, proceeding from left to right, the address/absolute address, PS, R0, R1, R2, R3, R4, R5, and the SP.

```
000544  000000  046076  000066  000000  000000  000000  000000  045316
000000 000 000 114 076 000 066 000 000 000 000 000 000 000 000 112 316
        ^@  ^@   >   L   6  ^@  ^@  ^@  ^@  ^@  ^@  ^@  ^@  ^@   N   J

000000  022646  000340  045770  000340  045770  000340  045770  000340
000000 045 246 000 340 113 370 000 340 113 370 000 340 113 370 000 340
         &   %      ^@       K      ^@       K      ^@       K      ^@
```

```
000020  045776  000340  011124  000340  045770  000340  050500  000340
000020 113 376 000 340 022 124 000 340 113 370 000 340 121 100 000 340
            K      ^@   T  ^R      ^@       K      ^@   @   Q      ^@

000040  000167  000543  000001  000001  000000  000000  000000  000353
000040 000 167 001 143 000 001 000 001 000 000 000 000 000 000 000 353
           ^@      ^A  ^A  ^@  ^A  ^@  ^@  ^@  ^@  ^@  ^@  ^@      ^@

000060  035444  000340  034034  000340  032776  000340  032402  000340
000060 073 044 000 340 070 034 000 340 065 376 000 340 065 002 000 340
        $   ;      ^@  ^\   8      ^@       5      ^@  ^B   5      ^@
```

3.4.1.7 **Crash Dump Analyzer Support Routine** - The Crash Dump Analyzer (CDA) support routine, when entered, prints the following message on a notification device specified at SYSGEN:

CRASH - CONT WITH SCRATCH MEDIA ON device mnemonic

You can then put the secondary crash dump device on line and depress the CONT switch on the operator's console. The Executive Crash Dump routine will dump memory to the crash dump device and halt the processor upon completion.

The procedure for subsequently invoking the Crash Dump Analyzer, which reads and formats the memory dump, is fully documented in the RSX-11M Crash Dump Analyzer Reference Manual.

3.4.2 Fault Isolation

Four causes can be identified when the system faults:

- A user-state task has faulted in such a way that it causes the system to fault.
- The user-written driver has faulted in such a way that it causes the system to fault.
- The RSX-11M system software itself has faulted.
- The host hardware has faulted.

When the system faults, you must immediately determine which of these four causes is responsible. This section presents some procedures that can help you isolate the source of the fault. Correcting the fault itself is your responsibility.

3.4.2.1 **Immediate Servicing** - Faults manifest themselves in roughly four ways (they are listed here in order of increasing difficulty of isolation):

1. If XDT is included, an unintended trap to XDT occurs.
2. The system displays text indicating a crash has occurred and halts.

3. The system halts but displays nothing.

4. The system is in an unintended loop.

The immediate aim, regardless of the fault manifestation, is to get to a point where you can obtain pertinent fault isolation data.

The following discusssions assume the existence of a system built with at least one of the debugging aids described in Section 3.4.1. (Note that the minimal system does not have space for these routines.)

Case 1--The system has trapped to XDT:

The trap may or may not be intended (for example, a previously set breakpoint). If it is not intended, type the X command, causing XDT to exit to location 40(octal), from which the Executive stack and register dump routine (if present), followed by either Panic Dump or the CDA support routine (if present), will be invoked. If, however, you have some idea of the source of the problem (for example, a recent coding change), then you can use XDT to examine pertinent data structures and code.

Case 2--The system has displayed text indicating a crash has occurred:

If the text consists of output from the Executive stack and register dump routine (refer to Section 3.4.1.1), all the basic information describing the state of the system has been displayed. If the text has been produced by the CDA support routine, follow the procedure for obtaining and formatting a memory dump as outlined in the RSX-11 Crash Dump Analyzer Reference Manual.

Case 3--The system has halted but displays no information:

Before taking any action, preserve the current PS and PC and the pertinent device registers (that is, examine and record the information these registers contain). The procedure depends on the particular PDP-11 processor. Consult the PDP-11 Processor Handbook for details.

After preserving the PS and PC, invoke your resident debugging aid: enter 40(octal) in the switch register, press LOAD ADDR, and then press START. The contents of 40(octal) cause the successive invocation of:

1. The Executive stack and register dump routine (if present)

2. Either Panic Dump or CDA support routine (if present)

Case 4--The system is in an unintended loop:

Proceed as follows:

1. Halt the processor.

2. Record the PC, the PS, and any pertinent device registers, as in case 3 above.

You may then want to step through a number of instructions in an attempt to locate the loop. For this attempt to be meaningful you must first disable the system clock. Proceed as follows: Examine the contents of word 777546 (if your system has a line-frequency clock) or word 772540 (if it has a programmable clock). Clear bit 6 in this word and redeposit the word.

NOTE

The system will not run until you have reenabled the clock.

After trying to locate the loop and reenabling the clock, transfer to location 40(octal), as in case 3.

3.4.2.2 **Pertinent Fault Isolation Data** - Before you attempt to locate the fault, you should dump system common (SYSCM). SYSCM contains a number of critical pointers and listheads. Find the appropriate limits for the module SYSCM by examining the Executive memory allocation map. Enter these limits to the Panic Dump routine or specify them in the command line used to invoke the Crash Dump Analyzer.

In addition, you should dump the dynamic storage region and the device tables. The dynamic storage region is the module INITL and any additional space gained from the SET /POOL command and the device tables are in SYSTB.

At this point, you have the following data:

- Processor Status Word (PSW)
- Program counter (PC)
- Stack
- R0 through R6
- Pertinent device registers
- Dynamic storage region (pool)
- Device tables
- System common

These data are the minimum required for effectively tracing the fault.

3.4.3 Fault Tracing

Three pointers in SYSCM are critical in fault tracing. These pointers are described below:

$STKDP - Stack Depth Indicator

This data item indicates which stack was being used at the time of the crash. $STKDP plays an important role in determining the origin of a fault. The following values apply:

+1--User (task-state) stack

0 or less--System stack

If the stack depth is +1, then the user has crashed the system. In a system with "brickwall" protection (for example, the mapped RSX-11M system), the nonprivileged user should not be able to crash the system.

$TKTCB - Pointer to the current Task Control Block (TCB)

This is the TCB of the user-level task in control of the CPU.

$HEADR - Pointer to the current task header.

The $HEADR word points to the header of the task currently running. The task header provides additional data to help isolate a fault. Figures 3-1 and 3-2 show the layout of task headers for unmapped and mapped systems, respectively.

The first word in the header is the user's stack pointer (SP) the last time it was saved. If the user branches wildly into the Executive, the Executive terminates the user task, but the system continues to function (possibly erroneously). Knowing the user's stack pointer provides one more link in the chain that may lead to the resolution of the fault.

The header (as pointed to by $HEADR) also contains the last-saved register set, just before the header guard word (the last word in the header--pointed to by H.GARD).

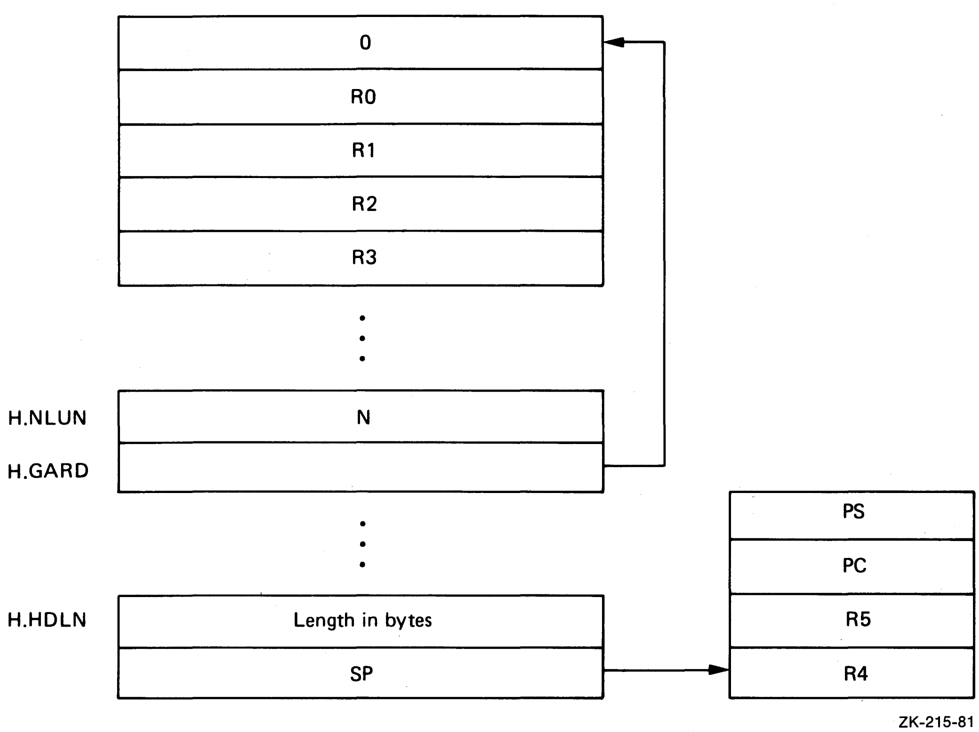


Figure 3-1 Task Header on an Unmapped System

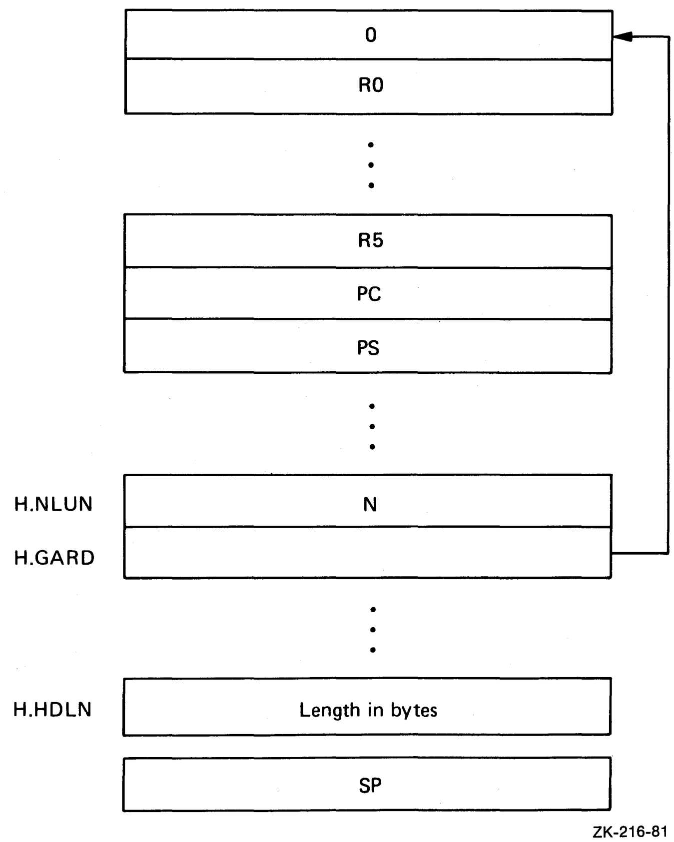
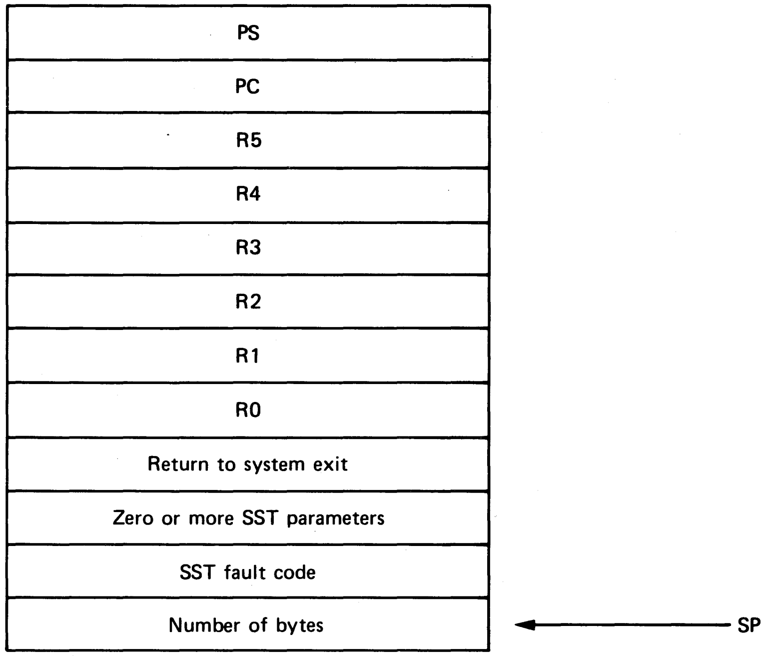


Figure 3-2 Task Header on a Mapped System

3.4.3.1 Tracing Faults Using the Executive Stack and Register Dump - To trace a fault after a display of the Executive stack and register contents, first examine the system stack pointer. Usually an Executive failure is the result of an SST-type trap within the Executive. If an SST does occur within the Executive, then the origin of the call on the crash-reporting routine is in the SST service module. (The crash call is initiated by issuing an IOT at a stack depth of zero or less.)

A call to crash also occurs in the Directive Dispatcher when an EMT is issued at a stack depth of zero or less, or a trap instruction is executed at a stack depth of less than zero. The stack structure in the case of an internal SST fault is shown in Figure 3-3.



ZK-217-81

Figure 3-3 Stack Structure: Internal SST Fault

The fault codes are:

```
0         ;ODD ADDRESS AND TRAPS TO 4
2         ;MEMORY PROTECT VIOLATION
4         ;BREAK POINT OR TRACE TRAP
6         ;IOT INSTRUCTION
10        ;ILLEGAL OR RESERVED INSTRUCTION
12        ;NON RSX EMT INSTRUCTION
14        ;TRAP INSTRUCTION
16        ;11/40 FLOATING POINT EXCEPTION
20        ;SST ABORT-BAD STACK
22        ;AST ABORT-BAD STACK
24        ;ABORT VIA DIRECTIVE
26        ;TASK LOAD READ FAILURE
30        ;TASK CHECKPOINT READ FAILURE
32        ;TASK EXIT WITH OUTSTANDING I/O
34        ;TASK MEMORY PARITY ERROR
```

The PC points to the instruction following the one that caused the SST failure. The number of bytes is the number normally transferred to the user stack when the particular type of SST occurs. If the number is 4, then a non-normal SST fault occurred, and only the PSW and PC are transferred. There are no SST parameters.

If the failure is detected in $DRDSP, the stack is the same as that shown in Figure 3-3, except that the number of bytes, the SST fault code (the fault codes are listed above), and the SST parameters are not present. The crash report message, however, will indicate that the failure occurred in $DRDSP.

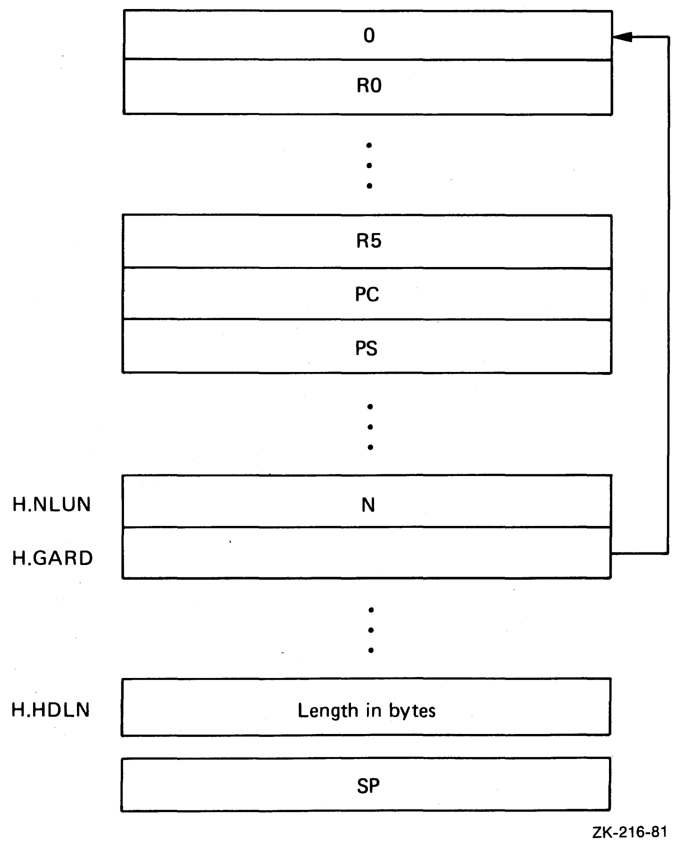
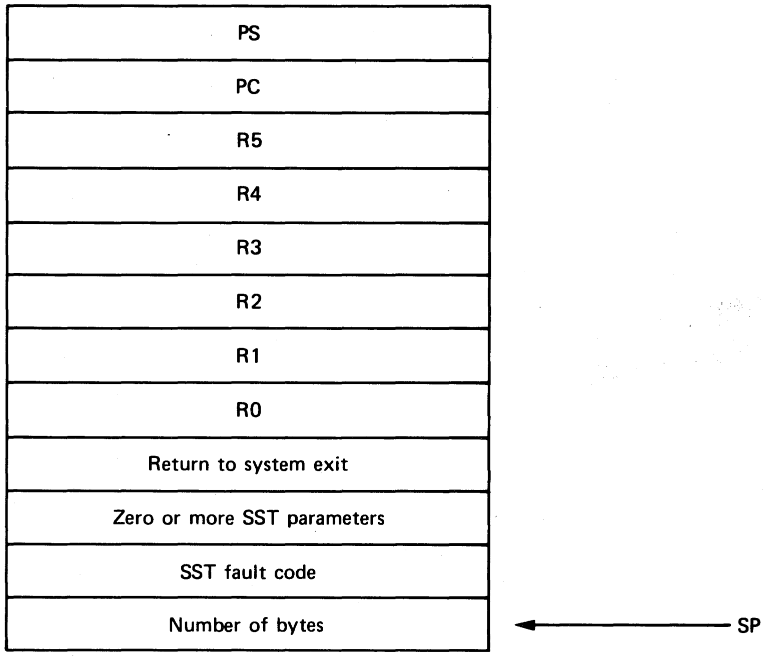


Figure 3-2 Task Header on a Mapped System

3.4.3.1 Tracing Faults Using the Executive Stack and Register Dump - To trace a fault after a display of the Executive stack and register contents, first examine the system stack pointer. Usually an Executive failure is the result of an SST-type trap within the Executive. If an SST does occur within the Executive, then the origin of the call on the crash-reporting routine is in the SST service module. (The crash call is initiated by issuing an IOT at a stack depth of zero or less.)

A call to crash also occurs in the Directive Dispatcher when an EMT is issued at a stack depth of zero or less, or a trap instruction is executed at a stack depth of less than zero. The stack structure in the case of an internal SST fault is shown in Figure 3-3.



ZK-217-81

Figure 3-3 Stack Structure: Internal SST Fault

The fault codes are:

0	;ODD ADDRESS AND TRAPS TO 4
2	;MEMORY PROTECT VIOLATION
4	;BREAK POINT OR TRACE TRAP
6	;IOT INSTRUCTION
10	;ILLEGAL OR RESERVED INSTRUCTION
12	;NON RSX EMT INSTRUCTION
14	;TRAP INSTRUCTION
16	;11/40 FLOATING POINT EXCEPTION
20	;SST ABORT-BAD STACK
22	;AST ABORT-BAD STACK
24	;ABORT VIA DIRECTIVE
26	;TASK LOAD READ FAILURE
30	;TASK CHECKPOINT READ FAILURE
32	;TASK EXIT WITH OUTSTANDING I/O
34	;TASK MEMORY PARITY ERROR

The PC points to the instruction following the one that caused the SST failure. The number of bytes is the number normally transferred to the user stack when the particular type of SST occurs. If the number is 4, then a non-normal SST fault occurred, and only the PSW and PC are transferred. There are no SST parameters.

If the failure is detected in $DRDSP, the stack is the same as that shown in Figure 3-3, except that the number of bytes, the SST fault code (the fault codes are listed above), and the SST parameters are not present. The crash report message, however, will indicate that the failure occurred in $DRDSP.

One SST-type failure, stack underflow, does not result in the stack structure of Figure 3-3. To determine where the crash occurred, first establish the stack structure; this can be deduced by the value of the SP and the contents of the top word on the stack. If the stack structure is that of Figure 3-3, then the failure occurred in $DRDSP, or was a normal SST crash. If the stack structure is that of Figure 3-4, then an abnormal SST crash has occurred.

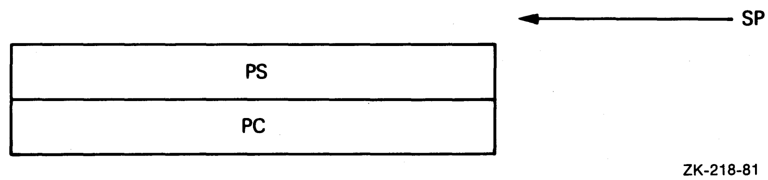


Figure 3-4 Stack Structure: Abnormal SST Fault

Abnormal SST failures occur when it is not possible to push information onto the stack without forcing another SST fault. When this situation occurs, a direct jump to the crash-reporting routine is made, rather than an IOT crash. The PS and PC on the stack are those of the actual crash, and the address printed out by the crash-reporting routine is the address of the fault rather than the address of the IOT that crashes the system. Note that the crash-reporting routine removes the PC and PSW of the IOT instruction from the stack, which in this case is incorrect. Thus, the SP appears to be four bytes greater than it really is (as in Figure 3-4).

You now have all the information needed to isolate the cause of the failure. From this point on, rely on personal experience and a knowledge of the interaction between the driver and the services provided by the Executive.

3.4.3.2 **Tracing Faults When the Processor Halts Without Display** - To trace a fault when the processor halts but displays no information (case 3 in Section 3.4.2.1 above), first examine $STKDP, $TKTCB, and $HEADR. The difficulty in tracing failures in this case is that the system stack is not directly associated with the cause of a failure.

By examining $STKDP, you can determine the system state at the time of failure. If it was in user state, the next step is to examine the user's stack. The examination focuses on scanning the stack for addresses that may be subroutine links that can ultimately lead to a thread of events isolating the fault. This is essentially the aim of looking at the system stack if $STKDP is zero or less.

Frequently, a fault can occur that causes the SP to point to the top of the stack plus 4. This fault results from issuing an RTI instruction. The top two items on the stack are data. The result is a wild branch and then, most probably, a halt. Figure 3-5 shows a case in which two data items are on the stack when the program executes an RTI instruction. The top of the stack points to a word containing 40100(octal). If location 40100(octal) contains a HALT instruction, the original SP is four bytes below the final SP, and fault tracing should begin from the original SP.

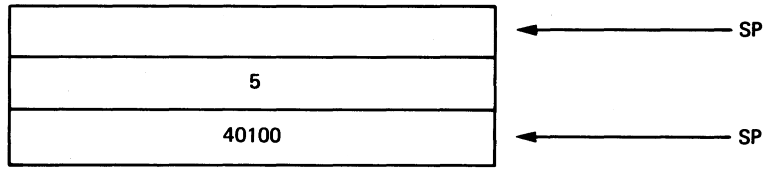


Figure 3-5 Stack Structure: Data Items on Stack

This type of fault also occurs when an RTS instruction is executed with an inconsistent stack. However, in that case, SP points to TOS+2.

A scan of the contents of the general registers may give some hint as to the neighborhood in which a fault (or the sequence of events leading up to the fault) occurred.

If the fault occurred in a new driver, a frequent source of clues is the buffer address and count words in the UCB (U.BUF, U.BUF+2, U.CNT), as are the activity flags (US.BSY and S.STS). Other locations in both the UCB and SCB may also provide information that may help locate the source of the fault.

3.4.3.3 **Tracing Faults After an Unintended Loop** - To trace a fault when an unintended loop has occurred, first halt the processor.

After you halt the processor, the same state exists as was discussed in Section 3.4.3.2. Follow the same tracing procedure described there. A specific suggestion is to check for a stack overflow loop. Patterns of data successively duplicated on the stack indicate a stack looping failure.

3.4.3.4 **Additional Hints for Tracing Faults** - Another item to check is the current (or last) I/O packet, the address of which is found in S.PKT of the SCB. The packet function (I.FCN) defines the last activity performed on the unit.

If trouble occurred in terminating an I/O request, a scan of the system dynamic memory region may provide some insight. This region starts at the address contained in $CRAVL, a cell in SYSCM. Because all I/O packets are built in system dynamic memory, their memory is returned to the dynamic memory region when they are successfully terminated. Following the link pointers in this region may reveal whether I/O completion proceeded to that point. In systems with QIO optimization, $PKAVL (SYSCM) points to a list of I/O packet-sized blocks of dynamic memory that are not linked into the $CRAVL chain.

A frequent error for an interrupt-driven device is to terminate an I/O packet twice when the device is not properly disabled on I/O completion and an unexpected interrupt occurs. This action ultimately produces a double deallocation of the same packet of dynamic memory. Double deallocation of a dynamic buffer in RSX-11M causes a loop in the module $DEACB on the next deallocation (of a block of higher address) after the second deallocation of the same block. At that time, R2 and R3 both contain the address of the I/O packet memory that has been doubly deallocated. If XDT has been included in the system, the deallocation routine checks for bad deallocation and crashes the system if it occurs.

3.4.4 Rebuilding and Reincorporating a Driver

The procedure for rebuilding and reincorporating a driver into your system depends on whether the driver is resident or loadable. The two subsections that follow describe the procedure for each kind of driver.

3.4.4.1 Rebuilding and Reincorporating a Resident Driver - The procedure for rebuilding and reincorporating a resident driver involves four steps:

NOTE

In the examples that follow:

- x (as in [11,2x] is equal to 0 for unmapped systems and 4 for mapped systems
- xxDRV is the name of the driver you are rebuilding or reincorporating

1. Correct and assemble the driver and/or device data structures.

   Assuming that the object system has been bootstrapped, appropriate volumes have been mounted, and the source code for the user driver and/or device data base has been updated, then the following commands effect the reassembly of both the driver and the data base:

   ```
   >SET /UIC=[11,2x](RET)
   >RUN $MAC(RET)
   MAC>xxDRV=LB:[1,1]EXEMC/ML,SY:[11,10]RSXMC,xxDRV(RET)
   MAC>USRTB=LB:[1,1]EXEMC/ML,SY:[11,10]RSXMC,USRTB(RET)
   MAC>(CTRL/Z)
   ```

2. Update the Executive object module library.

   After reassembling the user driver and/or data base, you must update the Executive object module library. The following commands will accomplish this:

   ```
   >SET /UIC=[1,2x](RET)
   >RUN $LBR(RET)
   LBR>LB:RSX11M/RP=[11,2X]xxDRV,USRTB(RET)
   LBR>(CTRL/Z)
   ```

3. Do Phase II of SYSGEN to rebuild the driver with the Executive.

4. Bootstrap the new system.

   The new system can now be bootstrapped with the MCR BOO command. If you are using the baseline system, first issue the following command:

   >INS BOO;-1(RET)

   Then issue the following command:

   >BOO [1,5x]RSX11M(RET)

3.4.4.2 **Rebuilding and Reincorporating a Loadable Driver** - A loadable driver is easier to reincorporate during debugging than a resident driver. After correcting and assembling the driver source, simply unload the old version, using the MCR UNL command, task-build the new one, and load it using the LOA command.

The data structure, once loaded, becomes a permanent part of the Executive. It is not removed by the UNL command. If the data structure is in error and cannot be patched, correct its source, reassemble, and task-build it. Then bootstrap the system before loading the corrected driver.

CHAPTER 4

WRITING AN I/O DRIVER--PROGRAMMING SPECIFICS

In Chapter 2, overviews were given for:

- Data structures
- Executive services
- Programming protocol

This chapter gives details for the data structures, and in addition discusses specifics of multicontroller drivers and the INTSV$ macro. Executive services are covered in Chapter 5. The protocol coverage in the discussion of programming protocol in Chapter 2 is detailed enough to make further elaboration unnecessary.

4.1 DATA STRUCTURES

The following elements in the I/O data structure are of concern to the programmer writing a driver:

- I/O packet
- DCB
- UCB
- SCB
- Device interrupt vector

The I/O data structure, and the first four control blocks listed above in particular, contain an abundance of data pertaining to input/output operations. Drivers themselves are involved with only a subset of the data.

In the detailed descriptions of the I/O packet, the DCB, the UCB, and the SCB that follow, most data fields (words or bytes) are classified by one of five descriptions. Two items in each description indicate:

- Whether the field is initialized in the data structure source
- What sort of access the driver has to the field during execution

The five descriptions are:

<initialized, not referenced>
Field is supplied in the data structure source code, and is not referenced by the driver during execution.

<initialized, read-only>
Field is supplied in the data structure source code, and may be read by the driver.

<not initialized, read-only>
Either an agent other than the driver establishes this field, or the driver sets it up once and thereafter references it read-only.

<not initialized, read-write>
Either the driver or some other agent establishes this field, and the driver may read it or write over it.

<not initialized, not referenced>
Field does not involve the driver in any way.

These five descriptions cover most of the fields in the four control blocks described in this section. Exceptions are noted in the text.

The final discussion in this section deals with the device interrupt vector.

4.1.1 The I/O Packet

Figure 4-1 shows the layout of the 18-word I/O packet, which is constructed and placed in the driver I/O queue by QIO directive processing, and is subsequently delivered to the driver by a call to $GTPKT. The DPB from which the I/O packet is generated is illustrated in Figure 4-2 (see Section 4.1.1.2).

4.1.1.1 **I/O Packet Details** - The I/O packet is built dynamically by QIO directive processing. Thus, no static fields exist with respect to a driver. I/O packets are created dynamically, and therefore the first two descriptions in Section 4.1 (<initialized, not referenced> and <initialized, read-only>) do not apply. Fields in the I/O packet (described below) are classified as not referenced, read-only, or read-write.

I.LNK

Driver access:

Not referenced.

Description:

Links I/O packets queued for a driver. A zero ends the chain. The listhead is in the SCB (S.LHD).

I.EFN

Driver access:

Not referenced.

Description:

Contains the event flag number as copied by QIO directive processing from the requester's DPB.

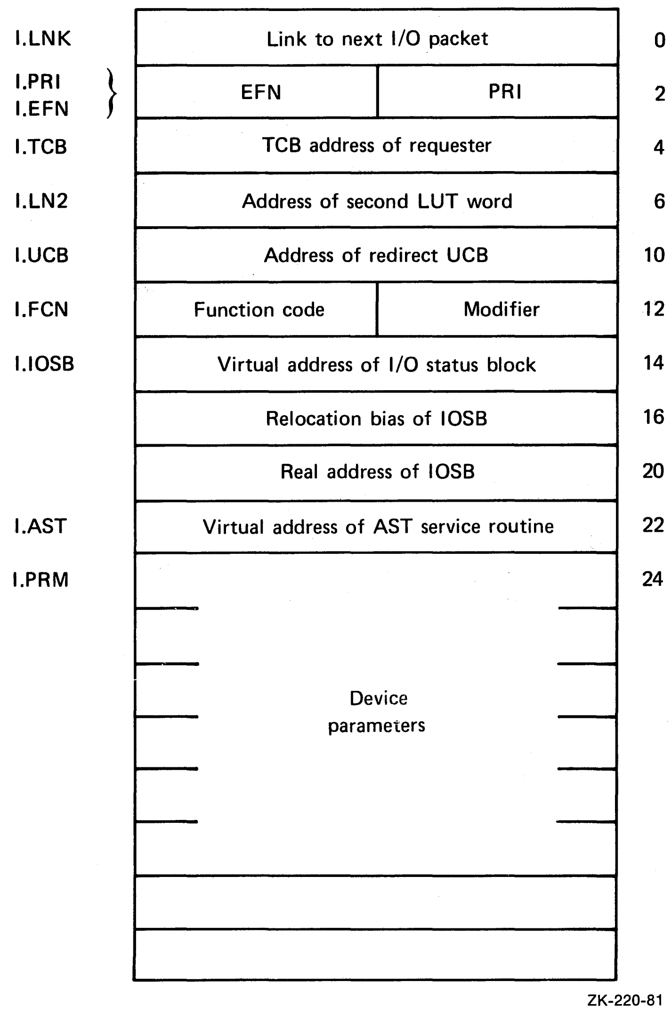


Figure 4-1 I/O Packet Format

I.PRI

Driver access:

Not referenced.

Description:

Priority copied from the TCB of the requesting task.

I.TCB

Driver access:

Read-only.

Description:

TCB address of the requesting task.

I.LN2

Driver access:

Not referenced.

Description:

Contains the address of the second word of the LUT entry in the task header to which the I/O request is directed. For open files on file-structured devices, this word contains the address of the window block; otherwise, it is zero.

I.UCB

Driver access:

Read-only.

Description:

Contains the address of the unit to which I/O is to be directed. I.UCB is the address of the Redirect UCB if the starting UCB has been subject to an MCR RED command. Used in cancel I/O routine to determine if the I/O request is from the task that is issuing the $IOKIL routine.

I.FCN

Driver access:

Read-only.

Description:

Contains the function code (see Table 4-1, Section 4.1.2.2) for the I/O request. The modifier byte is one or more subfunction bits that may be set.

I.IOSB

Driver access:

Not referenced.

Description:

I.IOSB contains the virtual address of the I/O Status Block (IOSB), if one is specified, or zero if one is not specified.

I.IOSB+2 and I.IOSB+4 contain the address doubleword for the IOSB (see Appendix A for a detailed description of the address doubleword). On an unmapped system, the first word is zero; the second word is the real address of the IOSB.

In a mapped system, the first word contains the relocation bias of the IOSB; the bias is, in effect, the number of the 32-word block in which the IOSB starts. This block number is derived by viewing available real memory as a collection of 32-word blocks numbered consecutively, starting with 0. Thus, if the IOSB starts at physical location 3210(octal), its block number is 32(octal).

The second word is formatted as follows:

Bits 0-5	Displacement in block (DIB)
Bits 6-12	All zeros
Bits 13-15	6

The displacement in block is the offset from the block base. In the above example, in which the IOSB starts at 3210(octal), the DIB is equal to 10(octal).

The value 6 in bits 13-15 is constant. It is used to cause an address reference through Kernel Address Page Register 6 (APR6). Again, see Appendix A for details.

Discussion of the address doubleword is deferred to an appendix because you seldom have to be concerned with its contents or format in writing a conventional driver. Its construction and subsequent manipulation are normally external to the driver. Subroutines are provided as Executive services for programmed I/O to render the manipulations of I/O transfers transparent to the driver itself.

I.AST

Driver access:

Not referenced.

Description:

Contains the virtual address of the AST service routine to be executed at I/O completion. If no address is specified, the field contains zero.

I.PRM

Driver access:

Not initialized, read-only.

Description:

Device-dependent parameters constructed from the last six words of the DPB. Note that if the I/O function is a transfer (refer to the description of D.MSK in Section 4.1.2.1), the buffer address (first DPB device-dependent parameter) is translated to an equivalent address doubleword. Therefore, device-dependent parameter n is copied to I.PRM +(2*(n-1))+2, where n is the number of the parameter and the first parameter is numbered P1.

4.1.1.2 **The QIO Directive Parameter Block (DPB)** - The QIO DPB is constructed as shown in Figure 4-2.

The parameters in the DPB have the following meanings.

Length (required):

The length of the DPB, which for the RSX-11M QIO directive is always fixed at 12(decimal) words.

DIC (required):

Directive Identification Code. For the QIO directive, this value is 1. For QIOW it is 3.

Function Code (required):

The code of the requested I/O function (0 through 31.).

Length	DIC	0
Function code	Modifier	2
Reserved	LUN	4
Priority	EFN	6
I/O status block address		10
AST address		12
Device-dependent parameters		14

ZK-221-81

Figure 4-2 QIO Directive Parameter Block (DPB)

Modifier:

Device-dependent modifier bits.

Reserved:

Reserved byte; must not be used.

LUN (required):

Logical Unit Number.

Priority:

Request priority. Ignored by RSX-11M.

EFN (optional):

Event flag number. Zero indicates no event flag. If you specify no event flag, QIOW$ directives are converted to QIO$ directives.

I/O Status Block Address (optional):

This word contains a pointer to the I/O status block, which is a 2-word, device-dependent, I/O-completion data packet formatted as:

Byte 0

I/O status byte.

Byte 1

Augmented data supplied by the driver.

Bytes 2 and 3

The contents of these bytes depend on the value of byte 0. If byte 0 equals 1, then these bytes usually contain the processed byte count. If byte 0 does not equal 0, then the contents are device-dependent.

AST Address (optional):

Address of an AST service routine.

Device Dependent Parameters:

Up to six parameters specific to the device and I/O function to be performed. Typically, for data transfer functions, the following four are used:

- Buffer address
- Byte count
- Carriage control type
- Logical block number

The fields for any optional parameters not specified will be filled with zeros.

4.1.2 The Device Control Block (DCB)

Figure 4-3 is a schematic layout of the DCB. The DCB describes the static characteristics of a device controller and the units attached to the controller. All fields must be specified except D.PCB, which is required only if you selected the loadable-driver option.

4.1.2.1 DCB Details - The fields in the DCB are described below:

D.LNK (link to next DCB)[1]

Driver access:

Initialized, not referenced.

Description:

Address link to the next DCB. A zero in this field indicates the last (or only) DCB in the chain.

D.UCB (pointer to first UCB)

Driver access:

Initialized, not referenced.

D.LNK	Link to next DCB (0=last)		0
D.UCB	Link to first UCB		2
D.NAM	Generic device name		4
D.UNIT	Highest unit no.	Lowest unit no.	6
D.UCBL	Length of UCB		10
D.DSP	Address of driver dispatch table		12
D.MSK	Legal function mask bits 0 - 15.		14
	Control function mask bits 0 - 15.		16
	No-op function mask bits 0 - 15.		20
	ACP function mask bits 0 - 15.		22
	Legal function mask bits 16. - 31.		24
	Control function mask bits 16. - 31.		26
	No-op function mask bits 16. - 31.		30
	ACP function mask bits 16. - 31.		32
D.PCB	Address of partition control block		34

ZK-222-81

Figure 4-3 Device Control Block

1. Parenthesized phrases indicate value to be initialized in the data base source code.

Description:

Address link to the U.DCB field of the first, and possibly the only, UCB associated with the DCB. For a given DCB, all UCBs are in contiguous memory locations and must all have the same length.

D.NAM (ASCII device name)

Driver access:

Initialized, not referenced.

Description:

Generic device name in ASCII by which device units are mnemonically referenced.

D.UNIT (unit number range)

Driver access:

Initialized, not referenced.

Description:

Unit number range for the device. This range covers those logical units available to the user for device assignment. Typically, the lowest number is zero, and the highest is n-1, where n is the number of device-units described by the DCB.

D.UCBL (UCB length)

Driver access:

Initialized, not referenced.

Description:

The UCB can have any length to meet the needs of the driver for variable storage. However, all UCBs for a given DCB must have the same length. The specified length must include the following prefix words if any or all are present:

- U.IOC
- U.ERHL
- U.ERHC
- U.ERSL
- U.ERSC
- U.LUIC
- U.OWN
- U.CLI
- U.MUP

D.DSP (dispatch table pointer)

Driver access:

Initialized, not referenced.

Description:

Address of the driver dispatch table.

When the Executive wishes to enter the driver at any of the four entry points contained in the driver dispatch table, it accesses D.DSP, locates the appropriate address in the table, and calls the driver at that address. A zero table address indicates that the (loadable) driver is not in memory. For a loadable driver, then, this field must be initialized to zero. If the driver does not process a given function, then this address points to a return instruction within the driver's code.

You must provide a driver dispatch table in the driver source. The label on this table is of the form $xxTBL; it must be a global label. The designation xx is the 2-character generic device name for the device. Thus, $TTTBL is the global label on the driver dispatch table for the generic device name TT. This table is an ordered, 4-word table containing the following entry points:

- I/O initiator
- Cancel I/O
- Device timeout
- Power failure

When a driver is entered at one of these entry points, entry conditions are as follows:

At initiator:

```
If UC.QUE=1
   R5 = UCB address
   R4 = SCB address
   R1 = Address of the I/O packet

If UC.QUE=0
   R5 = UCB address
```

Interrupts are allowed. (UC.QUE is a bit in U.CTL in the UCB. See Section 4.1.4.1.)

At cancel I/O:

```
R5 = UCB address
R4 = SCB address
R3 = Controller index
R1 = Address of TCB of current task
R0 = Address of active I/O packet
```

Device interrupts at or below the priority of the requesting device are locked out.

At device time-out:

R5 = UCB address
R4 = SCB address
R3 = Controller index
R2 = Address of device CSR
R0 = I/O status code IE.DNR (Device Not Ready)

Device interrupts at or below the priority of the requesting device are locked out.

At power failure:

R5 = UCB address
R4 = SCB address
R3 = Controller index

Interrupts are allowed. The power failure entry point of a loadable driver is called by LOA only for units that are on line and have UC.PWF set.

D.MSK (function masks)

Driver access:

Initialized, not referenced.

Description:

There are eight words, beginning at D.MSK, that are critical to the proper functioning of a device driver. The Executive uses these words to validate and dispatch the I/O request specified by a QIO directive. Four masks, with two words per mask, are described by the bit configurations that you establish for these words:

- Legal function mask
- Control function mask
- No-op function mask
- ACP function mask

The QIO directive allows for 32 possible I/O functions. The masks, as stated, are filters to determine validity and I/O requirements for the subject driver.

The Executive filters the function code in the I/O request through the four masks. The I/O function code is the high-order byte of the function parameter issued with the QIO directive. The decimal representation of that high-order byte is equivalent to the decimal bit number of the mask. If you want the function to be true in one of the four masks, you must set the bit in that mask in the position that numerically corresponds to the function code. For example, the code for IO.RVB is 21 (octal) and its decimal representation is 17. If you want IO.RVB to be true for a mask, you must set bit number 17(decimal) in the mask.

The masks are laid out in memory in two 4-word groups. Each 4-word group covers 16 function codes. The first 4 words cover the function codes 0-15; the second 4 words cover codes 16-31. Below is the layout used for the driver example in Section 6.2.2.

```
.WORD  140033     ;LEGAL FUNCTION MASK CODES 0-15.
.WORD  30         ;CONTROL FUNCTION MASK CODES 0-15.
.WORD  140000     ;NO-OP FUNCTION MASK CODES 0-15.
.WORD  0          ;ACP FUNCTION MASK CODES 0-15.
.WORD  5          ;LEGAL FUNCTION MASK CODES 16.-31.
.WORD  0          ;CONTROL FUNCTION MASK CODES 16.-31.
.WORD  1          ;NO-OP FUNCTION MASK CODES 16.-31.
.WORD  4          ;ACP FUNCTION MASK CODES 16.-31.
```

The mask words filter sequentially as follows:

Legal Function Mask:

Legal function values have the corresponding bit position in this word set to 1. Function codes that are not legal are rejected by QIO directive processing, which returns IE.IFC in the I/O status block, provided an IOSB address was specified.

Control Function Mask:

If any device-dependent data exists in the DPB, and this data does not require further checking by the QIO directive processor, the function is considered to be a control function. Such a function allows QIO directive processing to copy the DPB device-dependent data directly into the I/O Packet.

No-op Function Mask:

A no-op function is any function that is considered successful as soon as it is issued. If the function is a no-op, QIO directive processing immediately marks the request successful; no additional filtering occurs.

ACP Function Mask:

If a function code is legal but specifies neither a control function nor a no-op, then it specifies either an ACP function or a transfer function. If a function code requires intervention of an Ancillary Control Processor (ACP), the corresponding bit in the ACP function mask must be set. ACP function codes must have a value greater than 7.

In the specific case of read-write virtual functions, you have the option to set the corresponding mask bits. If the corresponding mask bits for a read-write virtual function are set, QIO directive processing recognizes that a file-oriented function is being requested to a non-file-structured device and converts the request to a read-write logical function.

This conversion is particularly useful. Consider a read-write virtual function to a specific device:

1. If the device is file structured and a file is open on the specified LUN, the block number specified is converted from a virtual block number in the file to a logical block number on the medium, and the request is queued to the driver as a read-write logical function.

2. If the device is file structured and no file is open on the specified LUN, then an error is returned and no further action is taken.

3. If the device is not file structured, then the request is simply transformed to a read-write logical function and is queued to the driver. (Specified block number is unchanged.)

Transfer Function Processing:

Finally, if the function is not an ACP function, then by default it is a transfer function. All transfer functions cause the QIO directive processor to check the specified buffer for legality (that is, inclusion within the address space of the requesting task) and proper alignment (word or byte). In addition, the processor checks the number of bytes being transferred for proper modulus (that is, nonzero and a proper multiple).

Creating Mask Words:

Creating function mask words involves five steps:

1. Establish the I/O functions available on the device for which driver support is to be provided.

2. Build the legal function mask: Check the standard RSX-11M function mask values in Table 4-1 for equivalencies. Only the IO.KIL function is mandatory. IO.ATT and IO.DET functions, if used, must have the RSX-11M system interpretation. DIGITAL suggests that functions having an RSX-11M system counterpart use the RSX-11M code, but this is required only when the device is to be used in conjunction with an ACP. From the supported function list in Table 4-1, you can build the two legal function mask words.

3. Build the control function mask by asking:

   Does this function carry a standard buffer address and byte count in the first two device-dependent parameter words?

   If it does not, then either it qualifies as a control function, or the driver itself must effect the checking and conversion of any addresses to the format required by the driver. See Section 6.3 for an example of a driver that does this. (Buffer addresses in standard format are automatically converted to address doubleword format.)

   Control functions are essentially those functions whose DPBs do not contain buffer addresses or counts.

4. Create the no-op function mask by deciding which legal functions are to be set as no-ops. Typically, for compatibility with File Control Services (FCS) or Record Management Services (RMS) on non-file-structured devices, the file access/deaccess functions are selected as legal functions, even though no specific action is required to access or deaccess a non-file-structured device; thus, the access/deaccess functions are set to be no-ops.

5. Finally, include the ACP functions Write Virtual Block and Read Virtual Block for those drivers that support both read and write. (Include only one related ACP function if the driver supports only read or write.)

D.PCB (Partition Control Block)

Driver access:

Initialized, not referenced.

Description:

Address of the driver's Partition Control Block (PCB). This word is present in the DCB if and only if the loadable-driver SYSGEN option has been selected. It must be initialized to zero. You can extend the DCB by adding words after D.PCB.

A PCB exists for every partition in a system. MCR creates a PCB when the SET /MAIN or SET /SUB commands are given. If a driver is loadable, its PCB describes the partition in which it resides.

The Executive uses D.PCB together with D.DSP (the address of the driver dispatch table) to determine whether a driver is loadable or resident, and, if loadable, whether it is in memory. Zero and nonzero values for these two pointers have the meanings shown in Table 4-1.

Table 4-1
D.PCB and D.DSP Bit Definitions

D.PCB: \ D.DSP:	= 0	≠ 0
= 0	Loadable driver, not in memory	Resident driver
≠ 0	(not possible)	Loadable driver, in memory

ZK-223-81

4.1.2.2 **Establishing I/O Function Masks** - Table 4-2 is supplied to assist you in determining the proper values to set in the function masks. The mask values are given for each I/O function used by DIGITAL-supplied drivers. The bit number allows you to determine which mask group to use: for bits numbered 0 through 15, use the mask value for a word in the first 4-word group; for bits numbered 16 through 31, use the mask value for a word in the second 4-word group.

Table 4-2
Mask Values for Standard I/O Functions

Bit #	Mask Value	Related Symbolic	I/O Function
0	1	IO.KIL	Cancel I/O
1	2	IO.WLB	Write Logical Block
2	4	IO.RLB	Read Logical Block
3	10	IO.ATT	Attach Device
4	20	IO.DET	Detach Device
5	40		General Device Control
6	100		General Device Control
7	200		General Device Control
8	400		Diagnostics
9	1000	IO.FNA	Find File in Directory
10	2000	IO.ULK	Unlock Block
11	4000	IO.RNA	Remove File from Directory
12	10000	IO.ENA	Enter File in Directory
13	20000	IO.ACR	Access File for Read
14	40000	IO.ACW	Access File for Read/Write
15	100000	IO.ACE	Access File for Read/Write/Extend
16	1	IO.DAC	Deaccess File
17	2	IO.RVB	Read Virtual Block
18	4	IO.WVB	Write Virtual Block
19	10	IO.EXT	Extend File
20	20	IO.CRE	Create File
21	40	IO.DEL	Mark File for Delete
22	100	IO.RAT	Read File Attributes
23	200	IO.WAT	Write File Attributes
24	400	IO.APC	ACP Control
25	1000		Unused
26	2000		Unused
27	4000		Unused
28	10000		Unused
29	20000		Unused
30	40000		Unused
31	100000		Unused

Of the function mask values listed in Table 4-2, only IO.KIL is mandatory and has a fixed interpretation. However, if IO.ATT and IO.DET are used, they must have the standard meaning. (Refer to the RSX-11M/M-PLUS I/O Drivers Reference Manual for a description of standard I/O functions.) If QIO directive processing encounters a function code of 3 or 4 and the code is not set to be a no-op, QIO$ assumes that these codes represent Attach Device and Detach Device, respectively. The other codes are suggested but not mandatory. You are free to establish all other function-code values on non-file-structured devices. The mask words must reflect the proper filtering process.

If a driver is being written for a file-structured device, the standard function mask values of Table 4-2 must be established.

Tables 4-3, 4-4, and 4-5 are guides to determining the proper bit masks for disks, tapes, and unit record devices (such as terminals, card readers, line printers, and paper tape punches/readers).

Table 4-3
Mask Word Bit Settings for Disk Drives

Bit #	RSX-11M		Related Symbolic
	ACP	non-ACP	
0	c	c	IO.KIL
1	t	t	IO.WLB
2	t	t	IO.RLB
3	c	c	IO.ATT
4	c	c	IO.DET
5			IO.STC
6			
7	sa		IO.CLN
8	sd		Diagnostic
9	a		IO.FNA
10	a		IO.ULK
11	a		IO.RNA
12	a		IO.ENA
13	a	n	IO.ACR
14	a	n	IO.ACW
15	a	n	IO.ACE
16	a	n	IO.DAC
17	a	a	IO.RVB
18	a	a	IO.WVB
19	a		IO.EXT
20	a		IO.CRE
21	a		IO.DEL
22	a		IO.RAT
23	a		IO.WAT
24	a		IO.APC
25			
26			
27			
28			
29			
30			
31			

t - transfer function, bit set only in legal function mask
c - control function, bit set in legal and control function masks
n - no-op function, bit set in legal and no-op function masks
a - ACP function, bit set in legal and ACP function masks
sa - special case, bit set only in ACP function mask, but not legal
sd - special case, bit set only if diagnostic support in system and driver

Table 4-4
Mask Word Bit Settings for Magnetic Tape Drives

Bit #	RSX-11M		Related Symbolic
	ACP	non-ACP	
0	c	c	IO.KIL
1	t	t	IO.WLB
2	t	t	IO.RLB
3	c	c	IO.ATT
4	c	c	IO.DET
5	c	c	IO.STC
6	c	c	
7	sa		IO.CLN
8	sd	sd	Diagnostic
9	a		IO.FNA
10			IO.ULK
11			IO.RNA
12	n		IO.ENA
13	a	n	IO.ACR
14	a	n	IO.ACW
15	a	n	IO.ACE
16	a	n	IO.DAC
17	a	a	IO.RVB
18	a	a	IO.WVB
19	a		IO.EXT
20	a		IO.CRE
21			IO.DEL
22	a		IO.RAT
23			IO.WAT
24	a		IO.APC
25			
26			
27			
28			
29			
30			
31			

t - transfer function, bit set only in legal function mask
c - control function, bit set in legal and control function masks
n - no-op function, bit set in legal and no-op function masks
a - ACP function, bit set in legal and ACP function masks
sa - special case, bit set only in ACP function mask, but not legal
sd - special case, bit set only if diagnostic support in system and driver

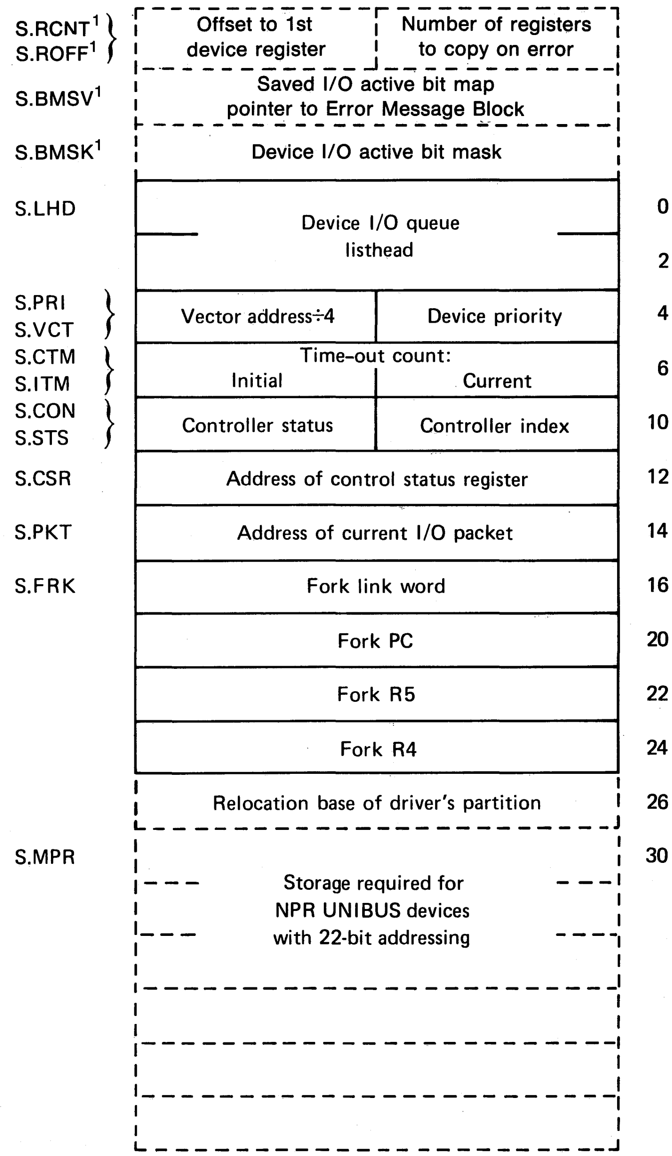
Table 4-5
Mask Word Bit Settings for Unit Record Devices

Bit #	RSX-11M ACP	RSX-11M non-ACP	Related Symbolic
0	c	c	IO.KIL
1	t	t	IO.WLB
2	t	t	IO.RLB
3	c	c	IO.ATT
4	c	c	IO.DET
5			IO.STC
6			
7	sa		IO.CLN
8	sd		Diagnostic
9	a		IO.FNA
10	a		IO.ULK
11	a		IO.RNA
12	a		IO.ENA
13	a	n	IO.ACR
14	a	n	IO.ACW
15	a	n	IO.ACE
16	a	n	IO.DAC
17	a	a	IO.RVB
18	a	a	IO.WVB
19	a		IO.EXT
20	a		IO.CRE
21	a		IO.DEL
22	a		IO.RAT
23	a		IO.WAT
24	a		IO.APC
25			
26			
27			
28			
29			
30			
31			

t - transfer function, bit set only in legal function mask
c - control function, bit set in legal and control function masks
n - no-op function, bit set in legal and no-op function masks
a - ACP function, bit set in legal and ACP function masks
sa - special case, bit set only in ACP function mask, but not legal
sd - special case, bit set only if diagnostic support in system and driver

4.1.3 The Status Control Block (SCB)

Figure 4-4 is a layout of the SCB. The SCB describes the status of a control unit that can run in parallel with all other control units.



1. These offsets exist for mass storage devices only, and only in systems incorporating error logging.

ZK-224-81

Figure 4-4 Status Control Block

4.1.3.1 **SCB Details** - The fields in the SCB are described below:

S.RCNT (used for error logging)

Driver Access:

Not initialized, not referenced.

Description:

The number of registers to copy on error. This offset exists for mass storage devices only (that is, when DV.MSD is set), and only in systems incorporating error logging.

S.ROFF (used for error logging)

Driver Access:

Not initialized, not referenced.

Description:

Offet to first device register. This offset exists for mass storage devices only (that is, when DV.MSD is set), and only in systems incorporating error logging.

S.BMSV (used for error logging)

Driver access:

Not initialized, not referenced.

Description:

Saved I/O active bit map and pointer to Error Message Block. This offset exists for mass storage devices only (that is, when DV.MSD is set), and only in systems incorporating error logging.

S.BMSK (used for error logging)

Driver access:

Not initialized, not referenced.

Description:

Device I/O active bit mask. This offset exists for mass storage devices only (that is, when DV.MSD is set), and only in systems incorporating error logging.

S.LHD (first word equals zero; second word points to first)[1]

Driver access:

Initialized, not referenced.

---

1. Parenthesized contents indicate values to be initialized in the data base source code.

Description:

Two words forming the I/O queue listhead. The first word points to the first I/O packet in the queue, and the second word points to the last I/O packet in the queue. If the queue is empty, the first word is zero, and the second word points to the first word.

S.PRI (device priority)

Driver access:

Initialized, read-only.

Description:

Contains the priority at which the device interrupts. Use symbolic values (for example, PR4) to initialize this field in your data base source. You define these symbolic values by issuing the HWDDF$ macro (refer to the sample data base in Section 6.2.1 and the listing of the HWDDF$ macro in Appendix C).

S.VCT (interrupt vector divided by 4)

Driver access:

Initialized, not referenced.

Description:

Interrupt vector address divided by 4. For loadable drivers, the MCR/VMR LOA function uses this field and the existence of driver symbol(s) $xxINT, $xxINP, and $xxOUT to initialize the device interrupt vector.

S.CTM (initialize to zero)

Driver access:

Not initialized, read-write.

Description:

RSX-11M supports device time-out, which enables a driver to limit the time that elapses between the issuing of an I/O operation and its termination. The current time-out count (in seconds) is initialized by moving S.ITM (initial time-out count) into S.CTM. The Executive clock service (in module TDSCH) examines active times, decrements them, and, if they reach zero, calls the driver at its device time-out entry point.

The internal clock count is kept in 1-second increments. Thus, a time count of 1 is not precise because the internal clocking mechanism is operating asynchronously with driver execution. The minimum meaningful clock interval is 2 if you intend to treat time-out as a consistently detectable error condition. Note, if the count is 0, that no time-out occurs; a zero value is, in fact, an indication that time-out is not operative. The maximum count is 255. You are responsible for setting this field. Resetting occurs at actual time-out or within $FORK.

S.ITM (set to initial time-out count)

Driver access:

Initialized, read-only.

Description:

Contains the initial time-out value.

S.CON (controller number times 2)

Driver access:

Initialized, read-only.

Description:

Controller number multiplied by 2. Drivers that are written to support more than one controller use this field. A driver may use S.CON to index into a controller table created and maintained internally by the driver itself. By indexing the controller table, the driver can service the correct controller when a device interrupts. See Section 4.2 for an example.

S.STS (initialize to zero)

Driver access:

Initialized, not referenced.

Description:

Establishes the controller as busy/not busy (nonzero/zero). This byte is the interlock mechanism for marking a driver as busy for a specific controller. The byte is tested and set by $GTPKT and reset by $IODON.

S.CSR (Control Status Register address)

Driver access:

Initialized, read-only.

Description:

Contains the address of the Control Status Register (CSR) for the device controller. The driver uses S.CSR to initiate I/O operations and to access, by indexing, other registers related to the device that are located in the I/O page. This address need not be a CSR; it need only be a member of the device's register set. It is accessed at system bootstrap time to determine if the interface exists on the system hosting the Executive. The Executive uses S.CSR to set the off-line bit at bootstrap so that system software can be interchanged between systems without an intervening system generation. The MCR LOA function also references S.CSR when it processes a loadable data base. Otherwise, only the driver itself accesses S.CSR.

S.PKT (reserve one word of storage)

Driver access:

Not initialized, read-only.

Description:

Address of the current I/O packet established by $GTPKT. The Executive uses this field to retrieve the I/O packet address upon the completion of an I/O request. S.PKT is not zeroed after the packet is completed.

S.FRK (reserve four or five words of storage)

Driver access:

Not initialized, not referenced.

Description:

The four words starting at S.FRK are used for fork block storage if and when the driver deems it necessary to establish itself as a fork process. Fork block storage preserves the state of the driver, which is restored when the driver regains control at fork level. This area is automatically used if the driver calls $FORK.

The fork block is five words long instead of four if two conditions are met:

1. Loadable drivers have been selected as a SYSGEN option.

2. The system is mapped.

If these conditions are met, and the fork block is five words long, you must not use the fork block for any other purpose. In other words, you cannot share the space reserved for the fork block; if you attempt to do so, you will destroy the loadable driver's relocation base. In addition, the 5-word fork block should always be part of the SCB if the two conditions above are met.

S.MPR (reserve six words of storage)

Driver access:

Initialized, read-only.

Description:

Drivers use the six words starting at S.MPR for non-processor request (NPR) devices attached to a processor with 22-bit addressing. See Appendix B for details on initializing S.MPR.

4.1.4 The Unit Control Block (UCB)

Figure 4-5 is a layout of the UCB (a variable-length control block). One UCB exists for each physical device-unit generated into a system configuration. For user-added drivers, this control block is defined as part of the source code for the driver data structure.

4.1.4.1 **UCB Details** - The fields in the UCB are described below. Note that the fields drawn with dotted outlines are not present in every UCB; their existence depends on physical device type, whether the system is generated with error logging, and whether you are running on a multiuser system. Refer to the individual descriptions of the offsets for details.

U.IOC

Driver access:

Not initialized, not referenced.

Description:

For mass storage devices only (that is, when DV.MSD is set), and only in systems incorporating error logging: the total number of QIOs issued to the device. (Initialized to zero when device is mounted.)

U.ERSL

Device access:

Not initialized, not referenced.

Description:

For mass storage devices only (that is, when DV.MSD is set), and only in systems incorporating error logging: the maximum number of soft errors that the error logger will log for the device. Note that error logging will stop if either of the limits specified (in U.ERHL or in U.ERSL) is exceeded.

U.ERHL

Driver access:

Not initialized, not referenced.

Description:

For mass storage devices only (that is, when DV.MSD is set), and only in systems incorporating error logging: the maximum number of hard errors that the error logger will log for the device. Note that error logging will stop if either of the limits specified (in U.ERHL or in U.ERSL) is exceeded.

U.ERSC

Driver access:

Not initialized, not referenced.

Description:

For mass storage devices only (that is, when DV.MSD is set), and only in systems incorporating error logging: the total number of soft errors logged on the device. (Initialized to zero when device is mounted.)

U.ERHC

Driver access:

Not initialized, not referenced.

Description:

For mass storage devices only (that is, when DV.MSD is set), and only in systems incorporating error logging: the total number of hard errors logged on the device. (Initialized to zero when device is mounted.)

NOTE

The following two symbolic names, U.MUP and U.CLI, both refer to the same absolute offset; use of the offset is dependent on system software configuration. Details regarding the use of each symbolic name are contained in the descriptions of U.MUP and U.CLI.

U.MUP

Driver access:

Not initialized, not referenced.

Description:

For terminal UCBs only, and only in multiuser systems that include alternate CLI support: bits 1 to 4 contain an index to a table which contains the address of CLI Parser Block (CPB) for the current CLI; the remaining bits are used for other terminal-specific features, and defined as follows:

UM.OVR	Override CLI indicator
UM.CLI	CLI indicator
UM.DSB	Terminal disabled because CLI eliminated
UM.NBR	No broadcast

U.CLI

Driver access:

Not initialized, not referenced.

Description:

For systems without alternate CLI support, but which do include multiuser protection: multiuser protection flag word.

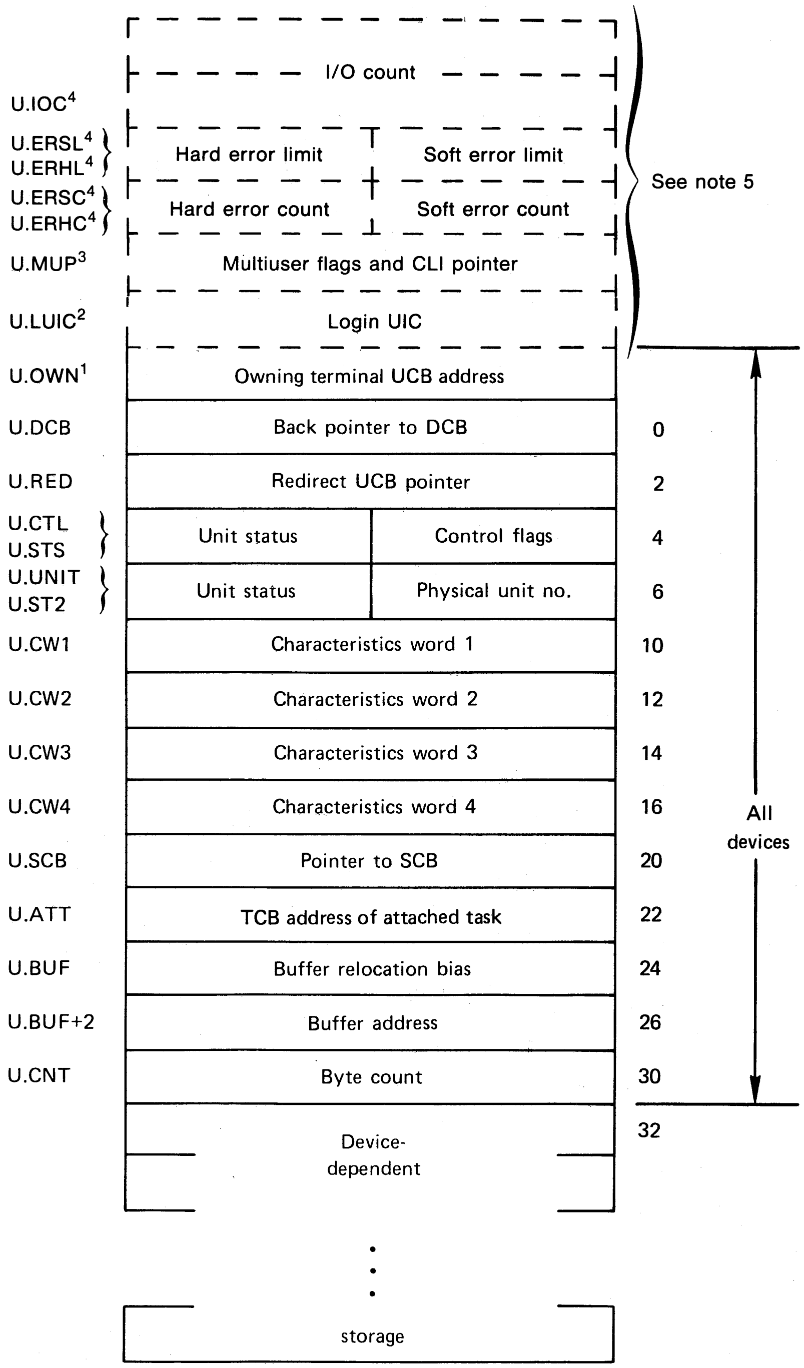
U.LUIC

Driver access:

Not initialized, not referenced.

Description:

For terminal UCBs only, and only in multiuser systems: the login UIC of the user at the particular terminal.



1. This offset appears only in multiuser systems.
2. These offsets appear only for terminal devices (that is, those devices that have DV.TTY set) in multiuser systems.
3. This offset appears only for terminal devices in multiuser systems that include alternate CLI support. In multiuser systems without alternate CLI support, this offset has a symbolic name of U.CLI. See descriptions and note in text.
4. These offsets appear only for mass storage devices (those devices that have DV.MSD set) in systems that employ error-logging.
5. These offsets are device-dependent.

ZK-225-81

Figure 4-5 Unit Control Block

U.OWN (initialize to zero)

Driver access:

Initialized, not referenced.

Description:

In multiuser systems only: the UCB address of the owning terminal for allocated devices.

U.DCB (pointer to associated DCB)

Driver access:

Initialized, not referenced.

Description:

Backpointer to the corresponding DCB. Because the UCB is a key control block in the I/O data structure, access to other control blocks usually occurs by means of links implanted in the UCB.

U.RED (redirect pointer--initialized to point to U.DCB of the UCB)

Driver access:

Initialized, not referenced.

Description:

Contains a pointer to the UCB to which this device-unit has been redirected. This field is updated as the result of an MCR Redirect command. The redirect chain ends when this word points to the beginning of the UCB itself (U.DCB of the UCB to be precise).

U.CTL (set by you)

Driver access:

Initialized, not referenced.

Description:

U.CTL and the function mask words in the DCB drive QIO directive processing. You are responsible for setting up this bit pattern. Any inaccuracy in the bit setting of U.CTL produces erroneous I/O processing. Bit symbols and their meanings are as follows:

UC.ALG - Alignment bit.

If this bit equals 0, then byte alignment of data buffers is allowed. If UC.ALG equals 1, then buffers must be aligned on word boundaries.

UC.ATT - Attach/Detach notification.

If this bit is set, then the driver is called when $GTPKT processes an Attach/Detach I/O function. Typically, the driver does not need to obtain control for Attach/Detach requests, and the Executive performs the entire function without any assistance from the driver.

UC.KIL - Unconditional Cancel I/O call bit.

If set, the driver is called on a Cancel I/O request, even if the unit specified is not busy. Typically, the driver is called on Cancel I/O only if an I/O operation is in progress.

UC.QUE - Queue bypass bit.

If set, the QIO directive processor calls the driver prior to queuing the I/O packet. After the processor makes this call, the driver is responsible for the disposition of the I/O packet. Typically, the processor queues an I/O packet prior to calling the driver, which later retrieves it by a call to $GTPKT.

UC.PWF - Unconditional call on power failure bit.

If set and the unit is on line, the driver is always to be called when the system is bootstrapped or power is restored after a power failure occurs. Typically, the driver is called on power restoration only when an I/O operation is in progress. Additionally, for loadable drivers, the driver is called when loaded if the unit is on line and UC.PWF is set.

UC.NPR - NPR device bit.

If set, the device is an NPR device. This bit determines the format of the 2-word address in U.BUF (details given in the discussion of U.BUF below).

UC.LGH - Buffer size mask bits (2 bits).

These two bits are used to check whether the byte count specified in an I/O request is a legal buffer modulus. You select one of the values below by ORing into the byte a 0, 1, or 3.

00 - Any buffer modulus valid
01 - Must have word alignment modulus
10 - Combination invalid
11 - Must have doubleword alignment modulus

UC.ALG and UC.LGH are independent settings.

UC.ATT, UC.KIL, UC.QUE, and UC.PWF are usually zero, especially for conventional drivers. Every driver, however, must be concerned with its particular values for UC.ALG, UC.NPR, and UC.LGH. You are totally responsible for the values in these bits, and erroneous values are likely to produce unpredictable results.

U.STS (initialize to zero)

Driver access:

Initialized, not referenced.

Description:

This byte contains device-independent status information. The bit meanings are as follows:

US.BSY - If set, device-unit is busy.

US.MNT - If set, volume is not mounted.

US.FOR - If set, volume is foreign.

US.MDM - If set, device is marked for dismount.

The unused bits in U.STS are reserved for system use and expansion. US.MDM, US.MNT, and US.FOR apply only to mountable devices.

U.UNIT (unit number)

Driver access:

Initialized, read-only.

Description:

This byte contains the physical unit number of the device-unit (that is, the value required for the hardware to access the specified drive unit). If the controller for the device supports only a single unit, the unit number is always zero.

U.ST2 (set by you)

Driver access:

Initialized, not referenced.

Description:

This byte contains additional device-independent status information. The bit meanings are as follows:

US.OFL - If set, the device is off line (that is, not in the configuration).

US.RED - If set, the device cannot be redirected.

US.PUB - If set, the device is a public device.

US.UMD - If set, the device is attached for diagnostics.

The remaining bits are reserved for system use and expansion.

U.CW1 (set by you)

Driver access:

Initialized, not referenced.

Description:

The first of a 4-word contiguous cluster of device characteristics information. U.CW1 and U.CW4 are device independent. U.CW2 and U.CW3 are device dependent. The four characteristics words are retrieved from the UCB and placed in the requester's buffer on issuance of a Get LUN information (GLUN$) Executive directive. It is your responsibility to supply the contents of these four words in the assembly source code of the driver's data structure.

U.CW1 is defined as follows (If a bit is set to 1, the corresponding characteristic is true for the device.):

DV.REC	Bit	0--Record-oriented device
DV.CCL	Bit	1--Carriage-control device
DV.TTY	Bit	2--Terminal device
DV.DIR	Bit	3--Directory device
DV.SDI	Bit	4--Single directory device
DV.SQD	Bit	5--Sequential device
DV.MSD	Bit	6--Mass storage device
DV.UMD	Bit	7--Device supports user-mode diagnostics
DV.EXT	Bit	8--Device attached to a 22-bit direct addressing controller
DV.SWL	Bit	9--Unit is software write-locked
DV.ISP	Bit	10--Input spooled device
DV.OSP	Bit	11--Output spooled device
DV.PSE	Bit	12--Pseudo device
DV.COM	Bit	13--Device mountable as a communications channel
DV.F11	Bit	14--Device mountable as a Files-11 device
DV.MNT	Bit	15--Device mountable

U.CW2 (initialize to zero)

Driver access:

Initialized, read-write.

Description:

Specific to a given device driver (available for working storage or constants).[1]

U.CW3 (initialize to zero)

Driver access:

Initialized, read-write.

Description:

Specific to a given device driver (available for working storage or constants).[1]

---

1. Exception: for block-structured devices, U.CW2 and U.CW3 cannot be used for working storage. In drivers for block-structured devices (disks and DECtape), these two words must be initialized to a double-precision number giving the total number of blocks on the device. Place the high-order bits in the low-order byte of U.CW2 and the low-order bits in U.CW3.

For tape UCBs, the high and low bytes of this word specify the highest and lowest densities, respectively, supported by the device. Symbolic names for U.CW3 are defined as follows for tape UCBs:

UN.UNS	Unsupported/unspecified
UD.200	200 bits/in, 7-track
UD.556	556 bits/in, 7-track
UD.800	800 bits/in, 7- or 9-track
UD.160	1600 bits/in, 9-track
UD.625	6250 bits/in, 9-track

For example, a TE16 tape drive supports densities of both 800 and 1600 bits/in. For a TE16 drive, U.CW3 would be coded as follows:

```
.BYTE   UD.800,UD.160
```

U.CW4 (set by you)

Driver access:

Initialized, read-only.

Description:

Default buffer size. This word is changed by the MCR command SET /BUF.

U.SCB (SCB pointer)

Driver access:

Initialized, read-only.

Description:

This field contains a pointer to the SCB for this UCB. In general, R4 contains the value in this word when the driver is entered by way of the driver dispatch table, because service routines frequently reference the SCB.

U.ATT (initialize to zero)

Driver access:

Initialized, not referenced.

Description:

If a task has attached itself to the device-unit, this field contains its TCB address.

U.BUF (reserve two words of storage)

Driver access:

Not initialized, read-write.

Description:

U.BUF labels two consecutive words that serve as a communication region between $GTPKT and the driver. U.BUF, U.BUF+2, and U.CNT receive the first three words from the I/O packet.

For transfer operations, the format of these two words depends on the setting of UC.NPR in U.CTL. The driver does not format the words; all formatting is completed before the driver receives control. For unmapped systems, the first word is zero, and the second word is the physical address of the buffer. For mapped systems, the format is determined by the UC.NPR bit, which is set for an NPR device and reset for a program transfer device.

The format for program transfer devices is identical to that for the second two words of I.IOSB in the I/O packet. See Section 4.1.1.1.

In general, the driver does not manipulate these words when performing I/O to a program transfer device. Instead, it uses the Executive routines Get Byte, Get Word, Put Byte, and Put Word to effect data transfers between the device and the user's buffer.

For NPR device drivers, the word layout is as follows:

Word 1

Bit 0	Go bit initially set to zero
Bits 1-3	Function code--set to zeros
Bits 4,5	Memory extension bits
Bits 6	Interrupt enable--set to zero
Bits 7-15	Zero

Word 2

Bits 0-15	Low-order 16-bits of physical address

It is your responsibility to set the function code, interrupt enable, and go bits. This action must be accomplished by a Bit Set (BIS) instruction so that the extension bits are not disturbed. The driver must move these words into the device control registers to initiate the I/O operation.

Note that when the system is unmapped, bits 4 and 5 are always zero, but this fact is transparent to the driver. Thus, NPR device drivers are not cognizant of the mapping state in the system.

The construction of U.BUF, U.BUF+2, and U.CNT occurs only if the requested function is a transfer function; if it is not, these three words contain the first three words of the I/O packet.

The details of the construction of the address doubleword appear in Appendix A.

U.CNT (reserve 1 word of storage)

Driver access:

Not initialized, read-write.

Description:

Contains the byte count of the buffer described by U.BUF. The driver uses this field in constructing the actual device request.

U.BUF and U.CNT keep track of the current data item in the buffer for the current transfer (except for NPR transfers). Because this field is being altered dynamically, the I/O packet may be needed to reissue an I/O operation, for instance, after a powerfail or error retry.

Device-Dependent Words:

Driver access:

Not initialized, read-write.

Description:

You establish this variable-length block of words to suit device-specific requirements.

4.1.5 The Device Interrupt Vector

For resident drivers only, the device interrupt vector must be initialized when defining data structures, and must not be altered dynamically. This practice makes the driver code independent of device register address assignments and of the actual location of the interrupt vector. The driver data structure must include a storage assignment and initialization for the interrupt vector with the priority set to PR7. See lines 81 through 85 in Section 6.2.1 (Section 6.2.1 contains the source code for the data structure of a sample resident driver).

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

4.2 MULTICONTROLLER DRIVERS

This section discusses the conditional code needed at the interrupt entry point of a driver that may handle one or several device controllers. This discussion leads to a description in the next section of the system macro INTSV$. INTSV$ contains multicontroller conditionals and other features to simplify interrupt entry coding.

Figure 4-6 shows the interrupt entry coding from the paper-tape-punch driver. This is an earlier version of the driver presented in its entirety in Section 6.2.2.

The coding is conditionalized on P$$P11-1. The symbol P$$P11 represents the number of controllers and is set at SYSGEN.

In a multicontroller device configuration, the controllers are numbered starting with 0. The code for a multicontroller driver contains a table (called CNTBL in the example in Figure 4-6) whose length in words is equal to the number of controllers. A number called the controller index--equal to the controller number times 2--is stored in the SCB for each controller, in byte S.CON.

When an I/O request occurs, and the driver is called at its initiator entry point, the driver first calls $GTPKT to obtain an I/O packet to process. Among the values returned by $GTPKT are the controller index (obtained from S.CON in the SCB) and the address of the UCB for the unit requesting I/O service.

The driver stores the requesting unit's UCB address in the controller table (CNTBl) at an offset equal to the controller index. Thus, for the driver at its interrupt entry point to access the requesting UCB, it needs simply to obtain the controller index.

The controller index is obtained from the controller number, which is encoded in the lowest four bits of the PS word in the device's interrupt vector. At its interrupt entry point the driver first saves the PS (line 9. in Figure 4-6), which was set from the device's interrupt vector upon interrupt. The PS must be saved with the first instruction of interrupt code because its lower four bits are the processor condition code bits, which generally change after each instruction is issued. Later, after the call to $INTSV, the driver constructs the controller index from the saved PS (lines 17.-19.). It then uses this index to obtain the UCB address (line 20.).

For single-controller devices, CNTBL is one word, TEMP is not needed to store the PS, and the UCB address is always the first (and only) entry in CNTBL.

NOTE

> The code sequence used in the following example is not valid for a loadable driver.

```
1 ;+
2 ; **-$PPINT-PC11 PAPER TAPE PUNCH CONTROLLER INTERRUPTS
3 ;-
4
5 $PPINT::                          ;;;REF LABEL
6
7        .IF GT  P$$P11-1
8
9        MOV     PS,TEMP            ;;;SAVE CONTROLLER NUMBER
10
11       .IFTF
12
13       CALL    $INTSV,PR4         ;;;SAVE REGISTERS AND SET PRIORITY
14
15       .IFT
16
17       MOV     TEMP,R4            ;;;RETRIEVE CONTROLLER NUMBER
18       BIC     #177760,R4         ;;;CLEAR ALL BUT CONTROLLER NUMBER
19       ASL     R4                 ;;;CONVERT TO CONTROLLER INDEX
20       MOV     CNTBL(R4),R5       ;;;RETRIEVE ADDRESS OF UCB
21
22       .IFF
23
24       MOV     CNTBL,R5           ;;;RETRIEVE ADDRESS OF UCB
25
26       .ENDC
```

Figure 4-6 Conditional Code for a Multicontroller Driver

4.3 THE INTSV$ MACRO

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

- Single or multiple controllers
- Loadable or resident drivers
- Mapped or unmapped systems

You can replace all the code in Figure 4-6 between lines 7 and 26 with the INTSV$ macro (as is done in the sample driver illustrated in Section 6.2.2). This is required for loadable drivers on mapped systems, because interrupts from hardware devices must be processed in kernel address space. In particular, the decoding of the PS word and the call to $INTSV must be done before entering the driver. Thus, a call to the Executive routine $INTSV within a loadable driver is illegal, and the MCR LOA function returns an error if loading is attempted.

When the INTSV$ macro is used for a loadable driver in a mapped system, the code from lines 9 to 19 inclusive (Figure 4-6) is not assembled as part of the driver. Instead, the LOA function allocates a block of dynamic memory in kernel address space to contain the interrupt coding. This block, called the Interrupt Control Block (ICB), also contains coding to perform the following:

1. Save the kernel mapping (APR5)
2. Load APR5 to map the driver
3. Transfer to the driver
4. Restore the mapping after return

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

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

4.3.1 Format

The format of the INTSV$ macro is:

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

xx

The 2-character device mnemonic.

pri

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

nctlr

The number of controllers the driver services.

pssave

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

ucbsave

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

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

R5 is the UCB address.

Example:

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

This usage of INTSV$ would effectively replace lines 7 through 26 in Figure 4-6. (P$$P11 is a symbol equated to the number of controllers.)

CHAPTER 5

EXECUTIVE SERVICES AVAILABLE TO I/O DRIVERS

This section contains the Executive routines typically used by I/O drivers. They are listed in alphabetical order. The descriptions are taken directly from the source code for the associated services.

We describe only the most widely used subroutines. Many other Executive service subroutines are available in modules IOSUB.MAC, EXESB.MAC, MEMAP.MAC, SYSXT.MAC, QUEUE.MAC, CORAL.MAC, and REQSB.MAC in UFD [11,10].

5.1 SYSTEM STATE REGISTER CONVENTIONS

In system state, R5 and R4 are, by convention, nonvolatile registers. This means that an internally called routine is required to save and restore these two registers if the routine destroys their contents. R3, R2, R1, and R0 are volatile registers and may be used by a called routine without save and restore responsibilities.

When a driver is entered directly from an interrupt, it is operating at interrupt level, not at system state. At interrupt level, any register the driver uses must be saved and restored. INTSV$ preserves R5 and R4 for the driver's use.

A routine may violate these conventions as long as an explicit statement exists in the program preface detailing the departure from conventions. Such departures should generally be avoided; they should be employed only when you can demonstrate that a departure from convention can improve overall system performance.

See D.DSP in Section 4.1.2.1 for the contents of registers when a driver is entered.

5.2 CONDITIONAL ROUTINES

Two of the routines ($GTWRD and $PTWRD) discussed in this chapter normally are assembled conditionally out of the Executive code. If a user-written driver requires either of these routines, the appropriate question must be answered affirmatively in the system generation dialog. See the descriptions of $GTWRD and $PTWRD below.

5.3 SERVICE CALLS

In the following descriptions, the file names mentioned are source modules found on the Executive source disk as [11,10]filnam.MAC.

$ACHKB/$ACHCK

ADDRESS CHECK

These routines are in the file EXESB. A driver may call either routine to address-check a task buffer while the task is the current task. The $ACHKB and $ACHCK routines are normally used only by drivers setting UC.QUE in U.CTL. See Section 6.3 for an example.

Calling sequences:

```
     CALL $ACHKB

  or

     CALL $ACHCK
```

Description:

```
;+
; **-$ACHKB-ADDRESS CHECK BYTE ALIGNED
; **-$ACHCK-ADDRESS CHECK WORD ALIGNED
;
; THIS ROUTINE IS CALLED TO ADDRESS CHECK A BLOCK OF MEMORY TO SEE
; WHETHER IT LIES WITHIN THE ADDRESS SPACE OF THE CURRENT TASK.
;
; INPUTS:
;
;       R0=STARTING ADDRESS OF THE BLOCK TO BE CHECKED.
;       R1=LENGTH OF THE BLOCK TO BE CHECKED IN BYTES.
;
; OUTPUTS:
;
;       C=1 IF ADDRESS CHECK FAILED.
;       C=0 IF ADDRESS CHECK SUCCEEDED.
;
;       R0 AND R3 ARE PRESERVED ACROSS CALL.
;-
```

$ALOCB

ALLOCATE CORE BUFFER

This routine is in the file CORAL.

Calling sequences:

    CALL $ALOCB

  or

    CALL $ALOC1

```
;+
; **-$ALOCB-ALLOCATE CORE BUFFER
; **-$ALOC1-ALLOCATE CORE BUFFER (ALTERNATE ENTRY)
;
; THIS ROUTINE IS CALLED TO ALLOCATE AN EXEC CORE BUFFER.  THE
; ALLOCATION ALGORITHM IS FIRST FIT AND BLOCKS ARE ALLOCATED IN
; MULTIPLES OF FOUR BYTES.
;
; INPUTS:
;
;       R0=ADDRESS OF CORE ALLOCATION LISTHEAD-2 IF ENTRY AT $ALOC1.
;       R1=SIZE OF THE CORE BUFFER TO ALLOCATE IN BYTES.
;
; OUTPUTS:
;
;       C=1 IF INSUFFICIENT CORE IS AVAILABLE TO ALLOCATE THE BLOCK.
;       C=0 IF THE BLOCK IS ALLOCATED.
;               R0=ADDRESS OF THE ALLOCATED BLOCK.
;               R1=LENGTH OF BLOCK ALLOCATED.
;-
```

$ASUMR

ASSIGN UNIBUS MAPPING REGISTERS

This routine is in the file MEMAP. It is used only for NPR devices requiring UNIBUS Mapping Registers, when 22-bit memory addressing is enabled. Normally, it is not called directly by an I/O driver. Rather, it is called from within the $STMAP routine. Refer to Appendix B for a discussion.

Calling sequence:

```
     CALL $ASUMR
```

Description:

```
;+
; **-$$ASUMR-ASSIGN UNIBUS MAPPING REGISTERS
;
; THIS ROUTINE IS CALLED TO ASSIGN A CONTIGUOUS SET OF UMR'S. NOTE THAT
; FOR THE SAKE OF SPEED, THE LINK WORD OF EACH MAPPING ASSIGNMENT BLOCK
; POINTS TO THE UMR ADDRESS (2ND) WORD OF THE BLOCK, NOT THE FIRST WORD.
; THE CURRENT STATE OF UMR ASSIGNMENT IS REPRESENTED BY A LINKED LIST OF
; MAPPING ASSIGNMENT BLOCKS, EACH BLOCK CONTAINING THE ADDRESS OF THE
; FIRST UMR ASSIGNED AND THE NUMBER OF UMR'S ASSIGNED TIMES 4. THE
; BLOCKS ARE LINKED IN THE ORDER OF INCREASING FIRST UMR ADDRESS.
;
; INPUTS:
;

;       R0=POINTER TO A MAPPING REGISTER ASSIGNMENT BLOCK.
;       M.UMRN(R0)=NUMBER OF UMR'S REQUIRED * 4.
;
; OUTPUTS:
;
;       ALL REGISTERS ARE PRESERVED.
;
;       C=0 IF THE UMR'S WERE SUCCESSFULLY ASSIGNED.
;               ALL FIELDS OF THE MAPPING REGISTER ASSIGNMENT BLOCK
;                       ARE INITIALIZED AND THE BLOCK IS LINKED INTO
;                       THE ASSIGNMENT LIST.
;       C=1 IF THE UMR'S COULD NOT BE ASSIGNED.
;-
```

$CLINS

CLOCK QUEUE INSERTION

This routine is in the file QUEUE.

Calling sequence:

    CALL $CLINS

Description:

```
;+
; **-$CLINS-CLOCK QUEUE INSERTION
;
; THIS ROUTINE IS CALLED TO MAKE AN ENTRY IN THE CLOCK QUEUE. THE ENTRY
; IS INSERTED SUCH THAT THE CLOCK QUEUE IS ORDERED IN ASCENDING TIME.
; THUS THE FRONT ENTRIES ARE MOST IMMINENT AND THE BACK LEAST.
;
; INPUTS:
;
;       R0=ADDRESS OF THE CLOCK QUEUE ENTRY CORE BLOCK.
;       R1=HIGH ORDER HALF OF DELTA TIME.
;       R2=LOW ORDER HALF OF DELTA TIME.
;       R4=REQUEST TYPE.
;       R5=ADDRESS OF REQUESTING TCB OR REQUEST IDENTIFIER.
;
; OUTPUTS:
;
;       THE CLOCK QUEUE ENTRY IS INSERTED IN THE CLOCK QUEUE ACCORDING
;       TO THE TIME THAT IT WILL COME DUE.
;-
```

$DEACB

DEALLOCATE CORE BUFFER

This routine is in the file CORAL.

Calling sequences:

```
     CALL $DEACB

  or

     CALL $DEAC1
```

```
;+
; **-$DEACB-DEALLOCATE CORE BUFFER
; **-$DEAC1-DEALLOCATE CORE BUFFER (ALTERNATE ENTRY)
;
; THIS ROUTINE IS CALLED TO DEALLOCATE AN EXEC CORE BUFFER.  THE BLOCK
; IS INSERTED INTO THE FREE BLOCK CHAIN BY CORE ADDRESS.  IF AN
; ADJACENT BLOCK IS CURRENTLY FREE, THEN THE TWO BLOCKS ARE MERGED
; AND INSERTED IN THE FREE BLOCK CHAIN.
;
; INPUTS:
;
;       R0=ADDRESS OF THE CORE BUFFER TO BE DEALLOCATED.
;       R1=SIZE OF THE CORE BUFFER TO DEALLOCATE IN BYTES.
;       R3=ADDRESS OF CORE ALLOCATION LISTHEAD-2 IF ENTRY AT $DEAC1.
;
; OUTPUTS:
;
;       THE CORE BLOCK IS MERGED INTO THE FREE CORE CHAIN BY CORE
;       ADDRESS AND IS MERGED IF NECESSARY WITH ADJACENT BLOCKS.
;-
```

$DEUMR

DEASSIGN UNIBUS MAPPING REGISTERS

This routine is in the file MEMAP. It is used only for NPR devices requiring UNIBUS Mapping Registers, when 22-bit memory addressing is enabled. Normally, it is not called directly by an I/O driver. Rather, it is called from within the $IODON routine. Refer to Appendix B for a discussion.

Calling sequence:

```
     CALL $DEUMR
```

Description:

```
;+
; **-$DEUMR-DEASSIGN UNIBUS MAPPING REGISTERS
;
; THIS ROUTINE IS CALLED TO DEASSIGN A CONTIGUOUS BLOCK OF UMR'S. IF
; THE MAPPING ASSIGNMENT BLOCK IS NOT IN THE LIST, NO ACTION IS TAKEN.
; NOTE THAT FOR THE SAKE OF ASSIGNMENT SPEED, THE LINK WORD POINTS TO
; THE UMR ADDRESS (2ND) WORD OF THE ASSIGNMENT BLOCK.
;
; INPUTS:
;       R2=POINTER TO ASSIGNMENT BLOCK.
;
; OUTPUTS:
;
;       R0 AND R1 ARE PRESERVED.
;-
```

$DVMSG

DEVICE MESSAGE OUTPUT

This routine is in file EXESB.

Calling sequence:

```
     CALL    $DVMSG
```

Description:

```
;+
; **-$DVMSG-DEVICE MESSAGE OUTPUT
;
; THIS ROUTINE IS CALLED TO SUBMIT A MESSAGE TO THE TASK TERMINATION
; NOTIFICATION TASK.  MESSAGES ARE EITHER DEVICE RELATED OR A
; CHECKPOINT WRITE FAILURE FROM THE LOADER.
;
; INPUTS:
;
;       R0=MESSAGE NUMBER.
;       R5=ADDRESS OF THE UCB OR TCB THAT THE MESSAGE APPLIES TO.
;
; OUTPUTS:
;
;       A FOUR WORD PACKET IS ALLOCATED, R0 AND R5 ARE STORED IN THE
;       SECOND AND THIRD WORDS, RESPECTIVELY, AND THE PACKET IS
;       THREADED INTO THE TASK TERMINATION NOTIFICATION TASK MESSAGE
;       QUEUE.
;
;       NOTE:  IF THE TASK TERMINATION NOTIFICATION TASK IS NOT
;                INSTALLED OR NO STORAGE CAN BE OBTAINED, THEN THE
;                MESSAGE REQUEST IS IGNORED.
;-
```

Note:

Drivers use only two codes in calling $DVMSG: T.NDNR (device not ready), and T.NDSE (select error). $DVMSG can be set up and called as follows:

```
MOV    #T.NDNR,R0
```

or

```
MOV    #T.NDSE,R0
CALL   $DVMSG
```

$EXRQP

EXECUTIVE REQUEST WITH QUEUE INSERT BY PRIORITY

This routine is in the file REQSB. $EXRQP requests the execution of a task after inserting a packet into the receive list of the task.

Calling sequence:

```
CALL $EXRQP
```

Description:

```
;+
; **-$EXRQP-EXECUTIVE REQUEST WITH QUEUE INSERT BY PRIORITY
; **-$EXRQF-EXECUTIVE REQUEST WITH QUEUE INSERT FIFO
; **-$EXRQN-EXECUTIVE REQUEST WITH NO QUEUE INSERTION
; **-$EXRQU-EXECUTIVE UNSTOP AND REQUEST WITH NO QUEUE INSERTION
; **-$EXRQS-EXECUTIVE REQUEST WITH NO CONDITIONAL SCHEDULE REQUEST
;
; THESE ROUTINES PROVIDE A STANDARD INTERFACE TO ALL TASKS REQUESTED BY
; THE EXECUTIVE
;
; INPUTS:
;
;       R0=TCB ADDRESS OF TASK TO REQUEST
;       R1=ADDR OF PACKET TO QUEUE TO TASK (IF ENTRY AT $EXRQP/$EXRQF)
;
; OUTPUTS:
;
;       C=0 IF THE REQUEST WAS SUCCESSFULLY COMPLETED.
;       C=1 IF THE TASK WAS NOT SUCCESSFULLY REQUESTED.
;            Z=0 IF PCB ALLOCATION FAILURE.
;            Z=1 IF TASK ACTIVE, BEING REMOVED, OR BEING FIXED.
;-
```

$FORK

FORK

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

Calling sequence:

```
    CALL $FORK
```

Description:

```
;+
; **-$FORK-FORK AND CREATE SYSTEM PROCESS
;
; THIS ROUTINE IS CALLED FROM AN I/O DRIVER TO CREATE A SYSTEM PROCESS THAT
; WILL RETURN TO THE DRIVER AT STACK DEPTH ZERO TO FINISH PROCESSING.
;
; INPUTS:
;
;         R5=ADDRESS OF THE UCB FOR THE UNIT BEING PROCESSED.
;
; OUTPUTS:
;
;         REGISTERS R5 AND R4 ARE SAVED IN THE CONTROLLER FORK BLOCK AND
;         A SYSTEM PROCESS IS CREATED. THE PROCESS IS LINKED TO THE FORK
;         QUEUE AND A JUMP TO $INTXT IS EXECUTED.
;-
```

Notes:

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

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

   If the clearing of the time-out count is not desired, a driver has two alternatives:

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

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

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

$FORK1

FORK1

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

Calling sequence:

```
CALL $FORK1
```

Description:

```
;+
; **-$FORK1-FORK AND CREATE SYSTEM PROCESS
;
; THIS ROUTINE IS AN ALTERNATE ENTRY TO CREATE A SYSTEM PROCESS AND
; SAVE REGISTER R5.
;
; INPUTS:
;
;       R4=ADDRESS OF THE LAST WORD OF A 3 WORD FORK BLOCK PLUS 2.
;       R5=REGISTER TO BE SAVED IN THE FORK BLOCK.
;
; OUTPUTS:
;
;       REGISTER R5 IS SAVED IN THE SPECIFIED FORK BLOCK AND A SYSTEM
;       PROCESS IS CREATED. THE PROCESS IS LINKED TO THE FORK QUEUE
;       AND A JUMP TO $INTXT IS EXECUTED.
;-
```

Notes:

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

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

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

$GTBYT

GET BYTE

This routine is in the file BFCTL. $GTBYT manipulates words U.BUF and U.BUF+2 in the UCB.

Calling sequence:

```
    CALL    $GTBYT
```

Description:

```
;+
; **-$GTBYT-GET NEXT BYTE FROM USER BUFFER
;
; THIS ROUTINE IS CALLED TO GET THE NEXT BYTE FROM THE USER BUFFER
; AND RETURN IT TO THE CALLER ON THE STACK. AFTER THE BYTE HAS BEEN
; FETCHED, THE NEXT BYTE ADDRESS IS INCREMENTED.
;
; INPUTS:
;
;       R5=ADDRESS OF THE UCB THAT CONTAINS THE BUFFER POINTERS.
;
; OUTPUTS:
;
;       THE NEXT BYTE IS FETCHED FROM THE USER BUFFER AND RETURNED
;       TO THE CALLER ON THE STACK. THE NEXT BYTE ADDRESS IS
;       INCREMENTED.
;
;       ALL REGISTERS ARE PRESERVED ACROSS CALL.
;-
```

$GTPKT

GET PACKET

This routine is in the file IOSUB.

Calling sequence:

```
    CALL    $GTPKT
```

Description:

```
;+
; **-$GTPKT-GET I/O PACKET FROM REQUEST QUEUE
;
; THIS ROUTINE IS CALLED BY DEVICE DRIVERS TO DEQUEUE THE NEXT I/O
; REQUEST TO PROCESS. IF THE DEVICE CONTROLLER IS BUSY, THEN A CARRY
; SET INDICATION IS RETURNED TO THE CALLER. ELSE AN ATTEMPT IS MADE TO
; DEQUEUE THE NEXT REQUEST FROM THE CONTROLLER QUEUE. IF NO REQUEST
; CAN BE DEQUEUED, THEN A CARRY SET INDICATION IS RETURNED TO THE
; CALLER. ELSE THE CONTROLLER IS SET BUSY AND A CARRY CLEAR
; INDICATION IS RETURNED TO THE CALLER.
;
; INPUTS:
;
;       R5=ADDRESS OF THE UCB OF THE CONTROLLER TO GET A PACKET FOR.
;
; OUTPUTS:
;
;       C=1 IF CONTROLLER IS BUSY OR NO REQUEST CAN BE DEQUEUED.
;       C=0 IF A REQUEST WAS SUCCESSFULLY DEQUEUED.
;               R1=ADDRESS OF THE I/O PACKET.
;               R2=PHYSICAL UNIT NUMBER.
;               R3=CONTROLLER INDEX.
;               R4=ADDRESS OF THE STATUS CONTROL BLOCK.
;               R5=ADDRESS OF THE UNIT CONTROL BLOCK.
;
;       NOTE: R4 AND R5 ARE DESTROYED BY THIS ROUTINE.
;-
```

$GTWRD

GET WORD

This routine is in the file BFCTL. $GTWRD manipulates words U.BUF and U.BUF+2 in the UCB, and is conditionally assembled. If your user-written driver requires this routine, answer Yes during Phase I of SYSGEN when the following question is asked:

*26. Include routine $GTWRD? [Y/N]:

If an LPA11 device (LA:) is included in your system, the $GTWRD routine is automatically included and Question 26 is not asked.

Calling sequence:

```
    CALL    $GTWRD
```

Description:

```
;+
; **-$GTWRD-GET NEXT WORD FROM USER BUFFER
;
; THIS ROUTINE IS CALLED TO GET THE NEXT WORD FROM THE USER BUFFER
; AND RETURN IT TO THE CALLER ON THE STACK. AFTER THE WORD HAS BEEN
; FETCHED, THE NEXT WORD ADDRESS IS CALCULATED.
;
; INPUTS:
;
;       R5=ADDRESS OF THE UCB THAT CONTAINS THE BUFFER POINTERS.
;
; OUTPUTS:
;
;       THE NEXT WORD IS FETCHED FROM THE USER BUFFER AND RETURNED
;       TO THE CALLER ON THE STACK. THE NEXT WORD ADDRESS IS CALCULATED.
;
;       ALL REGISTERS ARE PRESERVED ACROSS CALL.
;-
```

$INTSV

INTERRUPT SAVE

This routine is in the file SYSXT.

Calling sequence:

```
    CALL    $INTSV,PRn
```

n has a range of 0-7.

Description:

```
;+
; **-$INTSV-INTERRUPT SAVE
;
; THIS ROUTINE IS CALLED FROM AN INTERRUPT SERVICE ROUTINE WHEN AN
; INTERRUPT IS NOT GOING TO BE IMMEDIATELY DISMISSED. A SWITCH TO
; THE SYSTEM STACK IS EXECUTED IF THE CURRENT STACK DEPTH IS +1. WHEN
; THE INTERRUPT SERVICE ROUTINE FINISHES ITS PROCESSING, IT EITHER FORKS,
; JUMPS TO $INTXT, OR EXECUTES A RETURN.
;
; INPUTS:
;
;       4(SP)=PS WORD PUSHED BY INTERRUPT.
;       2(SP)=PC WORD PUSHED BY INTERRUPT.
;       0(SP)=SAVED R5 PUSHED BY 'JSR R5,$INTSV'.
;       0(R5)=NEW PROCESSOR PRIORITY.
;
; OUTPUTS:
;
;       REGISTER R4 IS PUSHED ONTO THE CURRENT STACK AND THE CURRENT
;       STACK DEPTH IS DECREMENTED. IF THE RESULT IS ZERO, THEN
;       A SWITCH TO THE SYSTEM STACK IS EXECUTED. THE NEW PROCESSOR
;       STATUS IS SET AND A CO-ROUTINE CALL TO THE CALLER IS EXECUTED.
;-
```

Note:

A system macro, INTSV$, is provided to simplify the coding of standard interrupt entry processing. See Section 4.3.

$INTXT

INTERRUPT EXIT

This routine is in the file SYSXT.

Calling sequence:

```
    JMP     $INTXT
```

or

```
    RETURN (if a call to $INTSV has been executed)
```

Description:

```
;+
; **-$INTXT-INTERRUPT EXIT
;
; THIS ROUTINE IS CALLED VIA A RETURN TO EXIT FROM AN INTERRUPT. IF THE
; STACK DEPTH IS NOT EQUAL TO ZERO, THEN REGISTERS R4 AND R5 ARE
; RESTORED AND AN RTI IS EXECUTED. ELSE A CHECK IS MADE TO SEE
; IF THERE ARE ANY ENTRIES IN THE FORK QUEUE. IF NONE, THEN R4 AND
; R5 ARE RESTORED AND AN RTI IS EXECUTED. ELSE REGISTERS R3 THRU
; R0 ARE SAVED ON THE CURRENT STACK AND A DIRECTIVE EXIT IS EXECUTED.
;
; INPUTS: (MAPPED SYSTEM)
;
;        06(SP)=PS WORD PUSHED BY INTERRUPT.
;        04(SP)=PC WORD PUSHED BY INTERRUPT.
;        02(SP)=SAVED R5.
;        00(SP)=SAVED R4.
;
; INPUTS: (REAL MEMORY SYSTEM)
;
;        NONE.
;-
```

$IOALT/$IODON

I/O DONE ALTERNATE ENTRY and I/O DONE

These routines are in the file IOSUB.

Calling sequences:

```
    CALL    $IOALT
    CALL    $IODON
```

Description:

```
;+
; **-$IOALT-I/O DONE (ALTERNATE ENTRY)
; **-$IODON-I/O DONE
;
; THIS ROUTINE IS CALLED BY DEVICE DRIVERS AT THE COMPLETION OF AN I/O REQUEST
; TO DO FINAL PROCESSING. THE UNIT AND CONTROLLER ARE SET IDLE AND $IOFIN IS
; ENTERED TO FINISH THE PROCESSING.
;
; INPUTS:
;
;       R0=FIRST I/O STATUS WORD.
;       R1=SECOND I/O STATUS WORD.
;       R2=STARTING AND FINAL ERROR RETRY COUNTS IF ERROR LOGGING DEVICE.
;       R5=ADDRESS OF THE UNIT CONTROL BLOCK OF THE UNIT BEING COMPLETED.
;
;       NOTE: IF ENTRY IS AT $IOALT, THEN R1 IS CLEARED TO SIGNIFY THAT THE
;             SECOND STATUS WORD IS ZERO.
;
; OUTPUTS:
;
;       THE UNIT AND CONTROLLER ARE SET IDLE.
;
;       R3=ADDRESS OF THE CURRENT I/O PACKET.
;-
```

NOTE

R4 is destroyed when either of these routines is called. The routines call $IOFIN, which destroys R4.

These routines push the address of routine $DQUMR onto the stack before returning to the driver. This precludes the use of the stack for temporary data storage by drivers when calling these routines.

$IOFIN

I/O FINISH

This routine is in the file IOSUB. Most drivers do not call $IOFIN, but you should be aware that this routine is executed when a driver calls $IOALT or $IODON. A driver that references an I/O packet before it is queued (bit UC.QUE set--see Section 6.3 for an example) calls $IOFIN if the driver finds an error while preprocessing the I/O packet.

Calling sequence:

```
    CALL    $IOFIN
```

Description:

```
;+
; **-$IOFIN-I/O FINISH
;
; THIS ROUTINE IS CALLED TO FINISH I/O PROCESSING IN CASES WHERE THE UNIT AND
; CONTROLLER ARE NOT TO BE DECLARED IDLE.
;
; INPUTS:
;
;       R0=FIRST I/O STATUS WORD.
;       R1=SECOND I/O STATUS WORD.
;       R3=ADDRESS OF THE I/O REQUEST PACKET.
;       R5=ADDRESS OF THE UNIT CONTROL BLOCK.
;
; OUTPUTS:
;
;       THE FOLLOWING ACTIONS ARE PERFORMED:
;
;       1-THE FINAL I/O STATUS VALUES ARE STORED IN THE I/O STATUS BLOCK IF
;               ONE WAS SPECIFIED.
;
;       2-THE I/O REQUEST COUNT IS DECREMENTED. IF THE RESULTANT COUNT IS
;               ZERO, THEN 'TS.RDN' IS CLEARED IN CASE THE TASK WAS
;               STOPPED FOR I/O RUNDOWN.
;
;       3-IF 'TS.CKR' IS SET, THEN IT IS CLEARED AND CHECKPOINTING OF THE
;               TASK IS INITIATED.
;
;       4-IF AN AST SERVICE ROUTINE WAS SPECIFIED, THEN AN AST IS QUEUED
;               FOR THE TASK. ELSE THE I/O PACKET IS DEALLOCATED.
;
;       5-A SIGNIFICANT EVENT OR EQUIVALENT IS DECLARED.
;
;       NOTE: R4 IS DESTROYED BY THIS ROUTINE.
;-
```

$MPUBM

MAP UNIBUS TO MEMORY

This routine is in the file MEMAP. $MPUBM is used only for NPR devices requiring UNIBUS Mapping Registers, when 22-bit memory addressing is enabled. See Appendix B for a discussion.

Calling sequence:

```
     CALL    $MPUBM
```

Description:

```
;+
; **-$MPUBM-MAP UNIBUS TO MEMORY
;
; THIS ROUTINE IS CALLED BY UNIBUS NPR DEVICE DRIVERS TO LOAD THE
; NECESSARY UNIBUS MAP REGISTERS TO EFFECT A TRANSFER TO MAIN MEM-
; ORY ON PDP-11 PROCESSORS WITH EXTENDED MEMORY.
;
; INPUTS:
;
;       R4=ADDRESS OF DEVICE SCB.
;       R5=ADDRESS OF DEVICE UCB.
;
; OUTPUTS:
;
;       THE UNIBUS MAP REGISTERS NECESSARY TO EFFECT THE  TRANSFER
;       ARE LOADED.
;
; NOTE: REGISTER R3 IS PRESERVED ACROSS CALL.
;-
```

$MPUB1

MAP UNIBUS TO MEMORY (ALTERNATE ENTRY)

This routine is in file MEMAP. It is used only for NPR devices that require UNIBUS Mapping Registers, support parallel operations, and have 22-bit memory addressing enabled. See Appendix B for a discussion of using this routine.

Calling sequence:

```
CALL   $MPUB1
```

Description:

```
;+
; **-$MPUB1-MAP UNIBUS TO MEMORY (ALTERNATE ENTRY)
;
; THIS ROUTINE IS CALLED BY UNIBUS NPR DEVICE DRIVERS TO LOAD THE
; NECESSARY UNIBUS MAP REGISTERS TO EFFECT A TRANSFER TO MAIN
; MEMORY ON PDP-11 PROCESSORS WITH EXTENDED MEMORY. THIS ALTERNATE
; ENTRY POINT ALLOWS THE DRIVER TO SPECIFY A NON-STANDARD UMR MAPPING
; ASSIGNMENT BLOCK.
;
; INPUTS:
;       R0=ADDRESS OF A UMR MAPPING ASSIGNMENT BLOCK
;       R4=ADDRESS OF DEVICE SCB
;       R5=ADDRESS OF DEVICE UCB
;
; OUTPUTS:
;
;       THE UNIBUS MAP REGISTERS NECESSARY TO EFFECT THE
;       TRANSFER ARE LOADED
;
; NOTE: REGISTER R3 IS PRESERVED ACROSS CALL.
;-
```

$PTBYT

PUT BYTE

This routine is in the file BFCTL. $PTBYT manipulates words U.BUF and U.BUF+2 in the UCB.

Calling sequence:

```
    CALL    $PTBYT
```

Description:

```
;+
; **-$PTBYT-PUT NEXT BYTE IN USER BUFFER
;
; THIS ROUTINE IS CALLED TO PUT A BYTE IN THE NEXT LOCATION IN
; USER BUFFER. AFTER THE BYTE HAS BEEN STORED, THE NEXT BYTE ADDRESS
; IS INCREMENTED.
;
; INPUTS:
;
;       R5=ADDRESS OF THE UCB THAT CONTAINS THE BUFFER POINTERS.
;       2(SP)=BYTE TO BE STORED IN THE NEXT LOCATION OF THE USER BUFFER.
;
; OUTPUTS:
;
;       THE BYTE IS STORED IN THE USER BUFFER AND REMOVED FROM
;       THE STACK. THE NEXT BYTE ADDRESS IS INCREMENTED.
;
;       ALL REGISTERS ARE PRESERVED ACROSS CALL.
;-
```

$PTWRD

PUT WORD

This routine is in the file BFCTL. $PTWRD manipulates words U.BUF and U.BUF+2 in the UCB. $PTWRD is conditionally assembled. If your user-written driver requires this routine, answer Yes during Phase I of SYSGEN to the following question:

    *27. Include routine $PTWRD? [Y/N]:

If an AD01 A/D controller device (AF:) or an AFC11 A/D controller device (AF:) is included in your system, the $PTWRD routine is automatically included and Question 27 is not asked.

    Do you intend to include a user written driver? [Y/N]:

Calling sequence:

```
CALL    $PTWRD
```

Description:

```
;+
; **-$PTWRD-PUT NEXT WORD IN USER BUFFER
;
; THIS ROUTINE IS CALLED TO PUT A WORD IN THE NEXT LOCATION IN
; USER BUFFER. AFTER THE WORD HAS BEEN STORED, THE NEXT WORD ADDRESS
; IS CALCULATED.
;
; INPUTS:
;
;       R5=ADDRESS OF THE UCB THAT CONTAINS THE BUFFER POINTERS.
;       2(SP)=WORD TO BE STORED IN THE NEXT LOCATION OF THE BUFFER.
;
; OUTPUTS:
;
;       THE WORD IS STORED IN THE USER BUFFER AND REMOVED FROM
;       THE STACK. THE NEXT WORD ADDRESS IS CALCULATED.
;
;       ALL REGISTERS ARE PRESERVED ACROSS CALL.
;-
```

$QINSP

QUEUE INSERTION BY PRIORITY

This routine is in the file QUEUE. A driver may call $QINSP to insert into the I/O queue an I/O packet that the Executive has not already placed in the queue. $QINSP is used only by drivers setting UC.QUE in U.CTL. See Section 6.3 for an example.

Calling sequence:

```
     CALL   $QINSP
```

Description:

```
;+
; **-$QINSP-QUEUE INSERTION BY PRIORITY
;
; THIS ROUTINE IS CALLED TO INSERT AN ENTRY IN A PRIORITY ORDERED
; LIST. THE LIST IS SEARCHED UNTIL AN ENTRY IS FOUND THAT HAS A
; LOWER PRIORITY OR THE END OF THE LIST IS REACHED. THE NEW
; ENTRY IS THEN LINKED INTO THE LIST AT THE APPROPRIATE POINT.
;
; INPUTS:
;
;       R0=ADDRESS OF THE TWO WORD LISTHEAD.
;       R1=ADDRESS OF THE ENTRY TO BE INSERTED.
;
;
; OUTPUTS:
;
;       THE ENTRY IS LINKED INTO THE LIST BY PRIORITY.
;
;       R0 AND R1 ARE PRESERVED ACROSS CALL.
;-
```

$QRMVF

QUEUE REMOVAL FROM FRONT OF LIST

This routine is in the file QUEUE.

Calling sequence:

    CALL $QRMVF

Description:

```
;+
; **-$QRMVG-QUEUE REMOVAL FROM FRONT OF LIST
;
; THIS ROUTINE IS CALLED TO REMOVE THE NEXT (FRONT) ENTRY FROM A
; LIST.  THE LIST ORGANIZATION MAY BE EITHER FIFO OR BY PRIORITY.
;
; INPUTS:
;
;       R0=ADDRESS OF THE TWO WORD LISTHEAD.
;
; OUTPUTS:
;
;       C=1 IF THERE ARE NO ENTRIES IN THE LIST.
;       C=0 IF THE NEXT ENTRY IS REMOVED FROM THE LIST.
;       R1=ADDRESS OF THE ENTRY REMOVED.
;
;       R0 IS PRESERVED ACROSS CALL.
;-
```

$RELOC

RELOCATE

Relocate is in the file MEMAP. A driver may call $RELOC to relocate a task virtual address while the task is the current task. Relocate is normally used only by drivers setting UC.QUE in U.CTL. See Section 6.3 for an example.

Calling Sequence:

```
     CALL    $RELOC
```

Description:

```
;+
; **-$RELOC-RELOCATE USER VIRTUAL ADDRESS
;
; THIS ROUTINE IS CALLED TO TRANSFORM A 16 BIT USER VIRTUAL ADDRESS
; INTO A RELOCATION BIAS AND DISPLACEMENT IN BLOCK RELATIVE TO APR6.
;
; INPUTS:
;
;       R0=USER VIRTUAL ADDRESS TO RELOCATE.
;
; OUTPUTS:
;
;       R1=RELOCATION BIAS TO BE LOADED INTO PAR6.
;       R2=DISPLACEMENT IN BLOCK PLUS 140000 (PAR6 BIAS).
;
;       R0 AND R3 ARE PRESERVED ACROSS CALL.
;-
```

$STMAP

SET UP UNIBUS MAPPING ADDRESS

This routine is in the file MEMAP. It is used only for NPR devices requiring UNIBUS Mapping Registers, when 22-bit memory addressing is enabled. See Appendix B for a discussion.

Calling sequence:

```
     CALL    $STMAP
```

Description:

```
;+
; **-$STMAP-SET UP UNIBUS MAPPING ADDRESS
;
; THIS ROUTINE IS CALLED BY UNIBUS NPR DEVICE DRIVERS TO SET UP THE
; UNIBUS MAPPING ADDRESS, FIRST ASSIGNING THE UMR'S. IF THE UMR'S
; CANNOT BE ALLOCATED, THE DRIVER'S MAPPING ASSIGNMENT BLOCK IS PLACED
; IN A WAIT QUEUE AND A RETURN TO THE DRIVER'S CALLER IS EXECUTED. THE
; ASSIGNMENT BLOCK WILL EVENTUALLY BE DEQUEUED WHEN THE UMR'S ARE
; AVAILABLE AND THE DRIVER WILL BE REMAPPED AND RETURNED TO WITH R1-R5
; PRESERVED AND THE NORMAL OUTPUTS OF THIS ROUTINE. THE DRIVER'S
; CONTEXT IS STORED IN THE ASSIGNMENT BLOCK AND FORK BLOCK WHILE IT IS
; BLOCKED AND IN THE WAIT QUEUE. ONCE A DRIVER'S MAPPING ASSIGNMENT
; BLOCK IS PLACED IN THE UMR WAIT QUEUE, IT IS NOT REMOVED FROM THE
; QUEUE UNTIL THE UMR'S ARE SUCCESSFULLY ASSIGNED. THIS STRATEGY
; ASSURES THAT WAITING DRIVERS WILL BE SERVICED FIFO AND THAT DRIVER'S
; WITH LARGE REQUESTS FOR UMR'S WILL NOT WAIT INDEFINITELY.
;
; INPUTS:
;
;       R4=ADDRESS OF DEVICE SCB.
;       R5=ADDRESS OF DEVICE UCB.
;       (SP)=RETURN TO DRIVER'S CALLER.
;
; OUTPUTS:
;
;       UNIBUS MAP ADDRESSES ARE SET UP IN THE DEVICE UCB AND THE
;       ACTUAL PHYSICAL ADDRESS IS MOVED TO THE SCB.
;
; NOTE: REGISTERS R1, R2, AND R3 ARE PRESERVED ACROSS CALL.
;-
```

NOTE

> This routine pushes the address of routine $DQUMR+2 onto the stack before returning to the caller. A driver, therefore, should not use the stack for temporary data storage when calling this routine.

$STMP1

SET UP UNIBUS MAPPING ADDRESS (ALTERNATE ENTRY)

This routine is in file MEMAP. It is used only for NPR devices that require UNIBUS Mapping Registers, support parallel operations, and have enabled 22-bit memory addressing. See Appendix B for a discussion of using this routine.

Calling sequence:

```
     CALL    $STMP1
```

Description:

```
;+
; **-$STMP1-SET UP UNIBUS MAPPING ADDRESS (ALTERNATE ENTRY)
;
; THIS ENTRY CODE SETS UP AN ALTERNATE DATA STRUCTURE USED AS
; A UMR MAPPING ASSIGNMENT BLOCK AND CONTEXT STORAGE BLOCK, IN
; THE SAME MANNER AS $STMAP USES THE FORK BLOCK AND MAPPING
; BLOCK IN THE SCB. THE FORMAT OF THE STRUCTURE IS AS FOLLOWS:
;
;        ---------------------          4 WORDS USED FOR SAVING
;        !                   !          DRIVER'S CONTEXT IN CASE
;        !                   !          UMR'S CAN'T BE MAPPED
;        !                   !          IMMEDIATELY.
;        !                   !
;        ---------------------
;        !                   !
;        !                   !          6 WORDS USED AS A UMR
;        !                   !          MAPPING ASSIGNMENT BLOCK.
;        !                   !
;        !                   !
;        !                   !
;        ---------------------
;
;
; INPUTS:
;
;        R0=ADDRESS OF THE DATA STRUCTURE DEPICTED ABOVE
;        R4=ADDRESS OF DEVICE SCB
;        R5=ADDRESS OF DEVICE UCB
;
; OUTPUTS:
;
;        DATA STRUCTURE POINTERS SET UP FOR ENTRY TO $STMP2 IN $STMAP
;
;-
```

NOTE

This routine pushes the address of routine $DQUMR+2 onto the stack before returning to the caller. A driver, therefore, should not use the stack for temporary data storage when calling this routine.

$SWSTK

SWITCH STACKS

This routine is in the file SYSXT.

Calling sequence:

    SWSTK$ label

The special macro in RSXMC.MAC must be used.

Description:

```
;+
; **$SWSTK-SWITCH STACKS
;
; THIS ROUTINE IS CALLED FROM TASK LEVEL TO SWITCH TO THE SYSTEM
; STACK THUS INHIBITING TASK SWITCHING.  THE CALLING TASK MUST BE
; PRIVILEGED IF RUNNING IN A MAPPED SYSTEM AND MAPPED TO THE EXEC.
; CONTROL IS PASSED HERE FROM DRDSP AFTER THE TRAP HAS OCCURRED AND
; $DIRSV HAS BEEN CALLED.
;
; CALLING SEQUENCE:
;
;       EMT   376   ;TRAP TO $EMSST IN DRDSP
;       .WORD ADDR   ;ADDRESS FOR RETURN TO USER STATE
;
; INPUTS AT THIS POINT:
;
;       R3=ADDRESS OF PC WORD OF TRAP ON STACK + 2
;
;       MAPPED SYSTEM:
;
;        22(SP)=PS PUSHED BY TRAP
;        20(SP)=PC PUSHED BY TRAP
;        16(SP)=SAVED R5
;        14(SP)=SAVED R4
;        12(SP)=SAVED R3
;        10(SP)=SAVED R2
;        06(SP)=SAVED R1
;        04(SP)=SAVED R0
;        02(SP)=RETURN ADDRESS FOR SYSTEM EXIT
;        00(SP)=104376
;
; UNMAPPED SYSTEM:
;
;        10(SP)=SAVED R3
;        06(SP)=SAVED R2
;        04(SP)=SAVED R1
;        02(SP)=SAVED R0
;        00(SP)=RETURN ADDRESS FOR SYSTEM EXIT
;
; OUTPUTS:
;
;       THE USER IS CALLED BACK ON THE SYSTEM STACK WITH ALL REGISTERS
;       PRESERVED.  TO RETURN TO TASK LEVEL THE CALLER MERELY EXECUTES
;       A RETURN.
;
;-
```

Note: Task registers are not modified.

CHAPTER 6

INCLUDING A USER-WRITTEN DRIVER--TWO EXAMPLES

The first example that follows illustrates the procedures required to add a resident driver and resident data base to an RSX-11M system so that they can run on a system without support for loadable drivers and without multiuser protection. The driver in the example supports the punch capability of the PC11 Paper Tape Reader/Punch.

Section 6.3 gives a coding example from a resident driver that inhibits the automatic packet queuing in QIO processing in order to address-check and relocate a special user buffer.

In addition to the examples shown in this chapter, you should review the source code for one or more standard DIGITAL-supplied drivers. Also, examine file SYSTB.MAC, which contains data structures created by SYSGEN.

6.1 DEVICE DESCRIPTION

The PC11 Paper Tape Reader/Punch is capable of reading 8-hole, unoiled, perforated paper tape at 300 char/s, and punching tape at 50 char/s. The system consists of a paper tape reader/punch and controller. A unit containing only a reader (PR11) is also available.

In reading tape, a set of photodiodes translates the presence or absence of holes in the tape to logic levels representing ones and zeroes. In punching tape, a mechanism translates logic levels representing ones and zeroes to the presence or absence, respectively, of holes in the tape. Any information read or punched is parallel-transferred through the controller. When an address is placed on the UNIBUS, the controller decodes the address and determines if the reader or punch has been selected. If one of the four device register addresses has been selected, the controller determines whether an input or an output operation should be performed. An input operation from the reader is initiated when the processor transmits a command to the paper tape reader status register. An output operation is initiated when the processor transfers a byte to the paper tape punch buffer register.

The controller enables the PDP-11 system to control the reading or punching of paper tape in a flexible manner. The reader can be operated independently of the punch; either device can be under direct program control or can operate without direct supervision (through the use of interrupts) so as to maintain continuous operation.

6.2 DATA BASE AND DRIVER SOURCE

The simplicity of writing a conventional driver for RSX-11M is obscured by the volume of explanation required to cover the universal case. As you will see below, building a conventional driver is a straightforward and modest undertaking.

6.2.1 The Data Base

The resident data base source shown below is self-explanatory. Take special note of the legal function mask words, starting at line 45. The standard function codes listed in Table 4-1 were used in creating the mask. Thus, the punch driver accepts the following I/O functions:

- Cancel I/O (CAN)
- Write Logical Block (WLB)
- Attach Device (ATT)
- Detach Device (DET)
- Access File For Read/Write (ACW)
- Access File For Read/Write/Extend (ACE)
- Deaccess File (DAC)
- Write Virtual Block (WVB)

The CAN function is mandatory. The WLB function is the only transfer function actually supported.

The ATT and DET functions are control functions. The three ATT/DET functions are legal for FCS and RMS compatibility, but are set to be no-ops. The WVB function is legal but is converted to WLB by QIO directive processing.

The bit mask for each function is as follows:

Function	Function Code(octal)	Mask(octal)	Bit Range(decimal)
CAN	0	000001	0-15.
WLB	1	000002	0-15.
ATT	3	000010	0-15.
DET	4	000020	0-15.
ACW	16	040000	0-15.
ACE	17	100000	0-15.
DAC	20	000001	16.-31.
WVB	22	000004	16.-31.

The legal masks result from adding the 0-15(decimal) bit-range words to form a mask and all the 16-31(decimal) bit-range words to form the second mask.

The control, no-op, and ACP masks are created in an analogous fashion, matching bit positions with legal function code meanings.

The complete set of mask words appears on lines 45 through 52 in the data structure source.

The function code selections for record-oriented devices are intended to match FCS and RMS requirements for file-structured devices. When FCS or RMS executes an Access For Write, it is simply marked as a no-op. This tends to minimize FCS and RMS device-dependent logic. Note also on line 84 that the controller number, which is encoded in the low byte of the interrupt vector PS word in RSX-11M, is set to zero. Finally, since the code represents a resident data base, note that lines 78 through 85 would be omitted for a loadable data base.

```
  1         .TITLE USRTB
  2         .IDENT /01/
  3
  4 ;
  5 ; COPYRIGHT 1976, DIGITAL EQUIPMENT CORP., MAYNARD, MASS.
  6 ;
  7 ; THIS SOFTWARE IS FURNISHED TO PURCHASER UNDER A LICENSE FOR USE
  8 ; ON A SINGLE COMPUTER SYSTEM AND CAN BE COPIED (WITH INCLUSION
  9 ; OF DEC'S COPYRIGHT NOTICE) ONLY FOR USE IN SUCH SYSTEM, EXCEPT
 10 ; AS MAY OTHERWISE BE PROVIDED IN WRITING BY DEC.
 11 ;
 12 ; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
 13 ; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
 14 ; EQUIPMENT CORPORATION.
 15 ;
 16 ; DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY
 17 ; OF ITS SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.
 18 ;
 19 ; VERSION 01
 20 ;
 21 ; T. J. PASCUSNIK  25-NOV-74
 22 ;
 23 ; CONTROL BLOCKS FOR PAPER TAPE PUNCH DRIVER
 24 ;
 25 ; MACRO LIBRARY CALLS
 26 ;
 27
 28          .MCALL DCBDF$,HWDDF$
 29          DCBDF$                    ;DEFINE DEVICE CONTROL BLOCK OFFSETS[1]
 30          HWDDF$                    ;DEFINE HARDWARE REGISTERS
 31
 32 ;
 33 ; PAPER TAPE PUNCH DEVICE DATA BASE
 34 ;
 35 ; PAPER TAPE PUNCH DEVICE CONTROL BLOCK
 36 ;
 37 $USRTB::
 38 PPDCB:   .WORD   0                 ;LINK TO NEXT DCB
 39          .WORD   .PP0              ;POINTER TO FIRST UCB
 40          .ASCII  /PP/              ;DEVICE NAME
 41          .BYTE   0,0               ;LOWEST AND HIGHEST UNIT NUMBERS COVERED
 42                                    ;   BY THIS DCB
 43          .WORD   PPND-PPST         ;LENGTH OF EACH UCB IN BYTES
 44          .WORD   $PPTBL            ;POINTER TO DRIVER DISPATCH TABLE
 45          .WORD   140033            ;LEGAL FUNCTION MASK CODES 0-15.
 46          .WORD   30                ;CONTROL FUNCTION MASK CODES 0-15.
```

1. Appendix C lists all macros that exist in RSX-11M to generate control block offsets.

```
        .WORD   140000          ;NO-P'ED FUNCTION MASK CODES 0-15.
        .WORD   0               ;ACP FUNCTION MASK CODES 0-15.
        .WORD   5               ;LEGAL FUNCTION MASK CODES 16.-31.
        .WORD   0               ;CONTROL FUNCTION MASK CODES 16.-31.
        .WORD   1               ;NO-OP'ED FUNCTION MASK CODES 16.-31.
        .WORD   4               ;ACP FUNCTION MASK CODES 16.-31.
;
; PAPER TAPE PUNCH UNIT CONTROL BLOCK
;
.PP0::
PPST=.
        .WORD   PPDCB           ;BACK POINTER TO DCB
        .WORD   .-2             ;POINTER TO REDIRECT UNIT UCB
        .BYTE   UC.ATT,0        ;CONTROL PROCESSING FLAG (PASS CONTROL
                                ;   ON ATTACH/DETACH), UNIT STATUS
        .BYTE   0,0             ;PHYSICAL UNIT NUMBER, UNIT STATUS EXTENSION
        .WORD   DV.REC          ;FIRST DEVICE CHARACTERISTICS WORD
                                ;   (RECORD-ORIENTED DEVICE)
        .WORD   0               ;SECOND DEVICE CHARACTERISTICS WORD
                                ;   (FOR INTERNAL USE BY DRIVER)
        .WORD   0               ;THIRD DEVICE CHARACTERISTICS WORD
                                ;   (FOR INTERNAL USE BY DRIVER)
        .WORD   64.             ;FOURTH DEVICE CHARACTERISTICS WORD
                                ;   (DEFAULT BUFFER SIZE IN BYTES)
        .WORD   PPSCB           ;POINTER TO SCB
        .WORD   0               ;TCB ADDRESS OF ATTACHED TASK
        .BLKW   1               ;RELOCATION BIAS OF BUFFER OF CURRENT
                                ;   I/O REQUEST
        .BLKW   1               ;ADDRESS OF BUFFER OF CURRENT I/O REQUEST
        .BLKW   1               ;BYTE COUNT OF CURRENT I/O REQUEST
PPND=.
;
; PAPER TAPE PUNCH INTERRUPT VECTOR
;
        .ASECT
.=74
        .WORD   $PPINT          ;ADDRESS OF INTERRUPT ROUTINE
        .WORD   PR7!0           ;INTERRUPT AT PRIORITY 7 (CONTROLLER=0)
        .PSECT
;
; PAPER TAPE PUNCH STATUS CONTROL BLOCK
;
PPSCB:  .WORD   0               ;CONTROLLER I/O QUEUE LISTHEAD
                                ;   (POINTER TO FIRST ENTRY)
        .WORD   .-2             ;   (POINTER TO LAST ENTRY)
        .BYTE   PR4,74/4        ;DEVICE PRI, INTERRUPT VECTOR ADDRESS/4
        .BYTE   0,4             ;CURRENT AND INITIAL TIMEOUT COUNTS
        .BYTE   0,0             ;CONTROLLER INDEX AND STATUS
                                ;   (0=IDLE, 1=BUSY)
        .WORD   177554          ;ADDRESS OF CONTROL STATUS REGISTER
        .BLKW   1               ;ADDRESS OF CURRENT I/O PACKET
        .BLKW   4               ;FORK BLOCK ALLOCATION

        .END
```

6.2.2 Driver Code

The code shown below for the punch capability of the PC11 is typical for a conventional driver. In fact, many of the descriptive comments can be used as a template and easily tailored to a driver for another device.

The structure of the driver follows the standard RSX-11M form, being separated into processing code for the following:

- Initiator
- Power failure
- Interrupt
- Time-out
- Cancel I/O

The driver itself services only Write Logical, Attach, and Detach I/O functions. Attach and Detach result in the punching of 170. nulls each for header and trailer.

Power failure and cancel I/O are handled by means of device time-out, as is the device-not-ready condition.

The driver uses the following Executive services:

$INTXT
$GTPKT
$GTBYT
$DVMSG

$INTSV is used indirectly; it is called by INTSV$ (line 165). See Section 4.3.

Comments beginning with ';;;' indicate that the instruction is being executed at a priority level greater than or equal to 4.

The code contained in lines 139-141 is used to inhibit the punching of a trailer on ATT/DET if the task is being aborted. This is especially desirable when the device is not ready (for example, out of paper tape) and the system has generated the DET function for the aborting process.

```
 1.    .TITLE  PPDRV
 2.    .IDENT  /02/
 3.
 4. ;
 5. ; COPYRIGHT   1976,  DIGITAL  EQUIPMENT  CORP.,  MAYNARD,   MASS.
 6. ;
 7. ; THIS SOFTWARE IS FURNISHED TO PURCHASER UNDER A LICENSE FOR USE
 8. ; ON A  SINGLE COMPUTER SYSTEM AND CAN BE COPIED (WITH INCLUSION
 9. ; OF DEC'S COPYRIGHT NOTICE) ONLY FOR USE IN SUCH SYSTEM, EXCEPT
10. ; AS MAY OTHERWISE BE PROVIDED IN WRITING BY DEC.
11. ;
12. ; THE  INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
13. ; NOTICE AND  SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
14. ; EQUIPMENT CORPORATION.
15. ;
16. ; DEC  ASSUMES NO  RESPONSIBILITY  FOR  THE  USE  OR RELIABILITY
17. ; OF ITS  SOFTWARE ON  EQUIPMENT WHICH  IS NOT SUPPLIED BY DEC.
18. ;
19. ; VERSION 02
20. ;
21. ; T. J. PASCUSNIK 25-NOV-74
22. ;
23. ; MODIFIED BY:
24. ;
```

```
; C. A. ANDERS 15-MAR-76
;
;       CA001 -- ADDITION OF LOADABLE DRIVER SUPPORT.
;
; T. J. PASCUSNIK 4-APR-76
;
;       TP031 -- EXECUTIVE DATA STRUCTURE CHANGES.
;
;
; PC11 PAPER TAPE PUNCH DRIVER
;
; MACRO LIBRARY CALLS
;

        .MCALL  ABODF$,HWDDF$,PKTDF$,TCBDF$
        ABODF$                  ;DEFINE TASK ABORT CODES
        HWDDF$                  ;DEFINE HARDWARE REGISTER SYMBOLS
        PKTDF$                  ;DEFINE I/O PACKET OFFSETS
        TCBDF$                  ;DEFINE TASK CONTROL BLOCK OFFSETS

;
; EQUATED SYMBOLS
;
; PAPER TAPE PUNCH STATUS WORD BIT DEFINITIONS (U.CW2)
;

  WAIT=100000                   ;WAITING FOR DEVICE TO COME ON-LINE (1=YES)
  ABORT=40000                   ;ABORT CURRENT I/O REQUEST (1=YES)
  TRAIL=200                     ;CURRENTLY PUNCHING TRAILER (1=YES)

;
; LOCAL DATA
;
; CONTROLLER IMPURE DATA TABLES (INDEXED BY CONTROLLER NUMBER)
;

  CNTBL:  .BLKW   P$$P11        ;ADDRESS OF UNIT CONTROL BLOCK


        .IF GT  P$$P11-1

  TEMP:   .BLKW   1             ;TEMPORARY STORAGE FOR CONTROLLER NUMBER

        .ENDC                           ; CA001
                                        ; CA001

;
; DRIVER DISPATCH TABLE
;

$PPTBL:: .WORD   PPINI          ;DEVICE INITIATOR ENTRY POINT
         .WORD   PPCAN          ;CANCEL I/O OPERATION ENTRY POINT
         .WORD   PPOUT          ;DEVICE TIMEOUT ENTRY POINT
         .WORD   PPPWF          ;POWERFAIL ENTRY POINT

;+
; **-PPINI-PC11 PAPER TAPE PUNCH CONTROLLER INITIATOR
;
; THIS ROUTINE IS ENTERED FROM THE QUEUE I/O DIRECTIVE WHEN AN I/O REQUEST
; IS QUEUED AND AT THE END OF A PREVIOUS I/O OPERATION TO PROPAGATE THE EXECU-
; TION OF THE DRIVER. IF THE SPECIFIED CONTROLLER IS NOT BUSY, THEN AN ATTEMPT
; IS MADE TO DEQUEUE THE NEXT I/O REQUEST. ELSE A RETURN TO THE CALLER IS
; EXECUTED. IF THE DEQUEUE ATTEMPT IS SUCCESSFUL, THEN THE NEXT I/O OPER-
; ATION IS INITIATED. A RETURN TO THE CALLER IS THEN EXECUTED.
```

```
;
; INPUTS:
;
; R5=ADDRESS OF THE UCB OF THE CONTROLLER TO BE INITIATED.
;
; OUTPUTS:
;
; IF THE SPECIFIED CONTROLLER IS NOT BUSY AND AN I/O REQUEST IS WAIT-
; ING TO BE PROCESSED, THEN THE REQUEST IS DEQUEUED AND THE I/O OPER-
; ATION IS INITIATED.
;-

        .ENABL  LSB
PPINI:  CALL    $GTPKT          ;GET AN I/O PACKET TO PROCESS
        BCS     PPPWF           ;IF CS CONTROLLER BUSY OR NO REQUEST

;
; THE FOLLOWING ARGUMENTS ARE RETURNED BY $GTPKT:
;
;       R1=ADDRESS OF THE I/O REQUEST PACKET.
;       R2=PHYSICAL UNIT NUMBER OF THE REQUEST UCB.
;       R3=CONTROLLER INDEX.
;       R4=ADDRESS OF THE STATUS CONTROL BLOCK.
;       R5=ADDRESS OF THE UCB OF THE CONTROLLER TO BE INITIATED.
;
; PAPER TAPE PUNCH I/O REQUEST PACKET FORMAT:
;
;       WD. 00 -- I/O QUEUE THREAD WORD.
;       WD. 01 -- REQUEST PRIORITY, EVENT FLAG NUMBER.
;       WD. 02 -- ADDRESS OF THE TCB OF THE REQUESTER TASK.
;       WD. 03 -- POINTER TO SECOND LUN WORD IN REQUESTER TASK HEADER.
;       WD. 04 -- CONTENTS OF THE FIRST LUN WORD IN REQUESTER TASK HEADER (UCB).
;       WD. 05 -- I/O FUNCTION CODE (IO.WLB, IO.ATT OR IO.DET).
;       WD. 06 -- VIRTUAL ADDRESS OF I/O STATUS BLOCK.
;       WD. 07 -- RELOCATION BIAS OF I/O STATUS BLOCK.
;       WD. 10 -- I/O STATUS BLOCK ADDRESS (REAL OR DISPLACEMENT + 140000).
;       WD. 11 -- VIRTUAL ADDRESS OF AST SERVICE ROUTINE.
;       WD. 12 -- RELOCATION BIAS OF I/O BUFFER.
;       WD. 13 -- BUFFER ADDRESS OF I/O TRANSFER.
;       WD. 14 -- NUMBER OF BYTES TO BE TRANSFERED.
;       WD. 15 -- NOT USED.
;       WD. 16 -- NOT USED.
;       WD. 17 -- NOT USED.
;       WD. 20 -- NOT USED.
;

        MOV     R5,CNTBL(R3)    ;SAVE UCB POINTER FOR INTERRUPT ROUTINE
        CLR     U.CW2(R5)       ;CLEAR ALL SWITCHES
        CMPB    I.FCN+1(R1),#IO.WLB/256. ;WRITE LOGICAL BLOCK FUNCTION?
        BEQ     10$             ;IF EQ YES
        MOV     I.TCB(R1),R0    ;GET REQUESTOR TCB ADDRESS
        BIT     #T2.ABO,T.ST2(R0) ;TASK BEING ABORTED?   ; TP031
        BNE     65$             ;IF NE YES - DON'T PUNCH TRAILER
        BIS     #TRAIL,U.CW2(R5) ;OTHERWISE FUNCTION IS ATTACH OR DETACH
                                ;    SET FLAG TO PUNCH TRAILER
        MOV     #170.,U.CNT(R5) ;SET COUNT FOR 170 NULLS
10$:    BIS     #WAIT,U.CW2(R5) ;ASSUME WAIT FOR DEVICE OFF LINE
        TST     @S.CSR(R4)      ;DEVICE OFF LINE?
        BMI     80$             ;IF MI YES
20$:    BIC     #WAIT,U.CW2(R5) ;DEVICE ON LINE, CLEAR WAIT CONDITION
        MOVB    S.ITM(R4),S.CTM(R4) ;SET TIMEOUT COUNT
        MOV     #100,@S.CSR(R4) ;ENABLE INTERRUPTS

```

```
;
; POWERFAIL IS HANDLED VIA THE DEVICE TIMEOUT FACILITY AND THEREFORE CAUSES
; NO IMMEDIATE ACTION ON THE DEVICE. THIS IS DONE TO AVOID A RACE CONDITION
; THAT COULD EXIST IN RESTARTING THE I/O OPERATION
;

  PPPWF:   RETURN

;+
; **-$PPINT-PC11 PAPER TAPE PUNCH CONTROLLER INTERUPTS
;-

$PPINT::                                 ;;;REF LABEL
           INTSV$  PP,PR4,P$$P11         ;;;GENERATE INTERRUPT SAVE CODE   ; CA001
           MOV     U.SCB(R5),R4          ;;;GET ADDRESS OF STATUS CONTROL BLOCK
           MOVB    S.ITM(R4),S.CTM(R4)   ;;;RESET TIMEOUT COUNT
           MOV     S.CSR(R4),R4          ;;;POINT R4 TO CONTROL STATUS REGISTER
           MOV     (R4)+,U.CW3(R5)       ;;;SAVE STATUS
           BMI     60$                   ;;;IF MI, ERROR
           SUB     #1,U.CNT(R5)          ;;;DECREMENT CHARACTER COUNT
           BCS     50$                   ;;;IF CS, THEN DONE
           TSTB    U.CW2(R5)             ;;;CURRENTLY PUNCHING TRAILER?
           BPL     30$                   ;;;IF PL NO
           CLRB    (R4)                  ;;;LOAD NULL INTO OUTPUT REGISTER
           BR      40$                   ;;;BRANCH TO LOAD IT
30$:       CALL    $GTBYT                ;;;GET NEXT BYTE FROM USER BUFFER
           MOVB    (SP)+,(R4)            ;;;LOAD BYTE INTO OUTPUT REGISTER
40$:       JMP     $INTXT                ;;;EXIT FROM INTERRUPT
50$:       INC     U.CNT(R5)             ;;;RESET BYTE COUNT
60$:       CLR     -(R4)                 ;;;DISABLE PUNCH INTERRUPTS
           CALL    $FORK                 ;;;CREATE SYSTEM PROCESS
           MOV     U.SCB(R5),R4          ;POINT R4 TO SCB
           MOV     S.PKT(R4),R1          ;POINT R1 TO I/O PACKET
           MOV     I.PRM+4(R1),R1        ;   AND PICK UP CHARACTER COUNT
           SUB     U.CNT(R5),R1          ;CALCULATE CHARACTERS TRANSFERRED
           MOV     #IS.SUC&377,R0        ;ASSUME SUCCESSFUL TRANSFER
           TST     U.CW3(R5)             ;DEVICE ERROR?
           BPL     70$                   ;IF PL NO
65$:       MOV     #IE.VER&377,R0        ;UNRECOVERABLE HARDWARE ERROR CODE
70$:       CALL    $IODON                ;INITIATE I/O COMPLETION
           BR      PPINI                 ;BRANCH BACK FOR NEXT REQUEST

;
; DEVICE TIMEOUT RESULTS IN A NOT READY MESSAGE BEING PUT OUT 4 TIMES A
; MINUTE. TIMEOUTS ARE CAUSED BY POWERFAILURE AND PUNCH FAULT CONDITIONS.
;

PPOUT:     CLRB    @S.CSR(R4)            ;;;DISABLE PUNCH INTERRUPT
           CLRB    PS                    ;;;ALLOW INTERRUPTS
80$:       MOV     #IE.DNR&377,R0        ;ASSUME DEVICE NOT READY ERROR
           MOV     U.CW2(R5),R1          ;ARE WE WAITING FOR DEVICE READY?
           BPL     70$                   ;IF PL NO, TERMINATE I/O REQUEST
           MOV     #IE.ABO&377,R0        ;ASSUME REQUEST IS TO BE ABORTED
           ASL     R1                    ;ABORT REQUEST?
           BMI     70$                   ;IF MI YES
           TST     @S.CSR(R4)            ;PUNCH READY?
           BPL     20$                   ;IF PL YES
           MOV     #T.NDNR,R0            ;SET FOR NOT READY MESSAGE
           MOVB    #1,S.CTM(R4)          ;SET TIMEOUT FOR 1 SECOND
           DECB    S.STS(R4)             ;TIME TO OUTPUT MESSAGE?
           BNE     PPPWF                 ;IF NE NO
           MOVB    #15.,S.STS(R4)        ;SET TO OUTPUT NEXT MESSAGE IN 15. SECONDS
           CALLR   $DVMSG                ;OUTPUT MESSAGE AND RETURN
           .DSABL  LSB
```

```

;
; CANCEL I/O OPERATION-FORCE I/O TO COMPLETE IF DEVICE IS NOT READY
;

PPCAN:     CMP      R1,I.TCB(R0)      ;;;REQUEST FOR CURRENT TASK?
           BNE      10$               ;;;IF NE NO
           BIS      #ABORT,U.CW2(R5)  ;;;SET FOR ABORT IF DEVICE NOT READY
10$:       RETURN                     ;;;

           .END
```

6.3 HANDLING SPECIAL USER BUFFERS

Some drivers need to handle user buffers in addition to the buffer that the Executive address-checks and relocates in a normal transfer request. Address checking and relocation operations must take place in the context of the task issuing the I/O request, because the mapping registers are set for the issuing task. However, in the normal driver interface, the task context after the call to $GTPKT is not, in general, that of the issuing task.

Thus, drivers that need to handle special buffers must be able to reference the I/O packet before it is queued, while the context of the issuing task is still intact.

The following coding excerpts from a standard RSX-11M driver (the AFC11 driver) illustrate the handling of a special user buffer. The key points are:

- The UC.QUE bit has been set in the control byte (U.CTL) of the UCB for each device/unit. (This is not shown in the coding excerpts below.)
- The routine that is referenced as the initiator entry point in the driver dispatch table performs the following actions:
  1. Picks up the user virtual address and conditionally address-checks it.
  2. Relocates the virtual address, storing the result back into the packet.
  3. Inserts the packet into the I/O queue and falls through to the entry point AFINI, which calls $GTPKT.
- The driver propagates its own execution by branching back to AFINI to call $GTPKT.

```
;
; DRIVER DISPATCH TABLE
;

$AFTBL::.WORD  AFCHK            ;DEVICE INITIATOR ENTRY POINT
        .WORD  AFCAN            ;CANCEL I/O OPERATION ENTRY POINT
        .WORD  AFOUT            ;DEVICE TIMEOUT ENTRY POINT
        .WORD  AFPWF            ;POWERFAIL ENTRY POINT
```

```
;+
; **-AFCHK-AFC11 ANALOG TO DIGITAL CONVERTER CONTROLLER PARAMETER CHECKING
;
; THIS ROUTINE IS ENTERED FROM THE QUEUE I/O DIRECTIVE WHEN AN I/O REQUEST
; IS RECEIVED FOR THE AFC11 ANALOG TO DIGITAL CONVERTOR. AFC11 I/O REQUESTS
; CONTAIN DEVICE DEPENDENT INFORMATION THAT MUST BE CHECKED IN THE CONTEXT
; OF THE ISSUING TASK. THEREFORE THE I/O REQUEST IS NOT QUEUED BEFORE CALLING
; THE DRIVER.
;
; INPUTS:
;
;       R1=ADDRESS OF THE I/O REQUEST PACKET.
;       R4=ADDRESS OF THE STATUS CONTROL BLOCK.
;       R5=ADDRESS OF THE UCB OF THE CONTROLER TO BE INITIATED.
;
; OUTPUTS:
;
;       THE CONTROL BUFFER IS ADDRESS CHECKED TO DETERMINE WHETHER IT LIES
;       WITHIN THE ISSUING TASK'S ADDRESS SPACE. IF THE ADDRESS CHECK
;       SUCCEEDS, THEN THE CONTROL BUFFER ADDRESS IS RELOCATED AND STORED
;       IN THE I/O PACKET, THE I/O PACKET IS INSERTED IN THE CONTROLLER
;       QUEUE, AND THE DEVICE INITIATOR IS ENTERED TO START THE CONTROLLER.
;       ELSE AN ILLEGAL BUFFER STATUS IS RETURNED AS THE FINAL I/O STATUS
;       OF THE REQUEST.
;-

AFCHK:  MOV     R1,R3               ;COPY ADDRESS OF I/O PACKET
        MOV     I.PRM+6(R3),R0      ;GET VIRTUAL ADDRESS OF CONTROL BUFFER

        .IF DF  A$$CHK!M$$MGE

        MOV     I.PRM+4(R3),R1      ;SET LENGTH OF BUFFER TO CHECK
        CALL    $ACHCK              ;ADDRESS CHECK CONTROL BUFFER
        BCC     10$                 ;IF CC ADDRESS OKAY
        MOV     #IE.SPC&377,R0      ;SET ILLEGAL BUFFER STATUS
        CALLR   $IOFIN              ;FINISH I/O OPERATION

        .ENDC

10$:    CALL    $RELOC              ;RELOCATE CONTROL BUFFER ADDRESS
        MOV     R1,I.PRM+6(R3)      ;SET RELOCATION BIAS OF CONTROL BUFFER
        MOV     R2,I.PRM+10(R3)     ;SET ADDRESS OF CONTROL BUFFER
        MOV     R3,R1               ;SET ADDRESS OF I/O PACKET
        MOV     R4,R0               ;SET ADDRESS OF I/O QUEUE LISTHEAD
        CALL    $QINSP              ;INSERT I/O PACKET IN REQUEST QUEUE
```

```
;+
; **-AFINI-AFC11 ANALOG TO DIGITAL CONVERTOR CONTROLLER INITIATOR
;
; THIS ROUTINE IS ENTERED FROM THE QUEUE I/O DIRECTIVE WHEN AN I/O REQUEST
; IS QUEUED AND AT THE END OF A PREVIOUS I/O OPERATION TO PROPAGATE THE EXECU-
; TION OF THE DRIVER. IF THE SPECIFIED CONTROLLER IS NOT BUSY, THEN AN ATTEMPT
; IS MADE TO DEQUE THE NEXT I/O REQUEST. ELSE A RETURN TO THE CALLER IS
; EXECUTED. IF THE DEQUEUE ATTEMPT IS SUCCESSFUL, THEN THE NEXT I/O OPER-
; ATION IS INITIATED. A RETURN TO THE CALLER IS THEN EXECUTED.
;
; INPUTS:
;
;       R5=ADDRESS OF THE UCB OF THE CONTROLLER TO BE INITIATED.
;
; OUTPUTS:
;
;       IF THE SPECIFIED CONTROLLER IS NOT BUSY AND AN I/O REQUEST IS WAIT
;       ING TO BE PROCESSED, THEN THE REQUEST IS DEQUEUED AND THE I/O OPER-
;       ATION IS INITIATED.
;-

        .ENABL  LSB
AFINI:  CALL    $GTPKT          ;GET AN I/O PACKET TO PROCESS
        BCS     AFCAN           ;IF CS CONTROLLER BUSY OR NO REQUEST
                                ;I/O CANCEL (AFCAN) IS A NO-OP FOR AFC11

;       .
;       .
;       .

        CALL    $IODON          ;FINISH I/O OPERATION
        BR      AFINI           ;GO AGAIN

;       .
;       .
;       .
```

APPENDIX A

DEVELOPMENT OF THE ADDRESS DOUBLEWORD

A.1 INTRODUCTION

You can generate an RSX-11M system as a mapped or an unmapped system. Mapped systems can accommodate configurations whose maximum physical memory is 4096K bytes. Individual tasks, however, are limited to 64K bytes. The addressing in a mapped system uses virtual addresses and memory mapping hardware. I/O transfers, however, use physical addresses 18 bits in length. Since the PDP-11 word size is 16 bits, some scheme is necessary to represent an address internally until it is actually used in an I/O operation. The choice was made to encode two words as the internal representation of a physical address, and to transform virtual addresses for I/O operations into the internal doubleword format.

A.2 CREATING THE ADDRESS DOUBLEWORD

For unmapped systems, the doubleword is simply a word of zeros followed by a word containing the real address.

On receipt of a QIO directive for mapped systems, the buffer address in the DPB, which contains a task virtual address, is converted to address doubleword format.

The virtual address in the DPB is structured as follows:

Bits 0-5	Displacement in terms of 32-word blocks
Bits 6-12.	Block number
Bits 13.-15.	Page Address Register (PAR) number

DEVELOPMENT OF THE ADDRESS DOUBLEWORD

The internal RSX-11M translation restructures this virtual address into an address doubleword as follows:

1. The relocation base contained in the PAR specified by the PAR number in the virtual address in the DPB is added to the block number in the virtual address. The result becomes the first word of the address doubleword. It represents the nth 32-word block in a memory viewed as a collection of 32-word blocks. Note that at the time the address doubleword is computed, the user issuing the QIO directive is mapped into the processor's memory management registers.

2. The second word is formed by placing the displacement-in-block (bits 0-5 of virtual address) into bits 0-5. The block number field was accommodated in the first word and bits 6-12. are cleared. Finally, a 6 is placed in bits 13.-15. to enable use of PAR #6, which the Executive uses to service I/O for program transfer devices.

   For non-processor request (NPR) devices, the driver requirements for manipulating the address doubleword are direct and are discussed with the description of U.BUF in Section 4.1.4.1.

APPENDIX B

DRIVERS FOR NPR DEVICES USING EXTENDED MEMORY

You must build special features into drivers for nonextended memory NPR devices attached to a PDP-11 processor with extended-memory support (22-bit addressing).

Nonextended memory NPR devices on the PDP-11 processor must perform I/O transfers by means of UNIBUS Mapping Registers (UMRs), as described in the PDP-11 Processor Handbook. One UMR is required for each 4K words involved in the transfer--as specified by the contents of U.CNT in the UCB. When multiple UMRs are required for a transfer, they must be contiguous.

A driver can be assigned UMRs through one of three procedures. These procedures involve the following:

1. Dynamically allocating UMRs for the duration of the data transfer

2. Dynamically allocating UMRs for longer periods of time

3. Statically allocating UMRs during system generation

NOTE

In large systems, using the second and third procedures above to hold UMRs for longer periods than necessary can result in the blocking of other drivers and a reduction in system throughput.

B.1 CALLING $STMAP AND $MPUBM OR $STMP1 AND $MPUB1

To obtain UMRs through use of the $STMAP and $MPUBM or $STMP1 and $MPUB1 routines, a driver must:

1. If it uses $STMAP and $MPUBM or $STMP1 and $MPUB1, allocate six additional words for a mapping register assignment block at the end of the device's SCB (at S.MPR). If it uses $STMP1 and $MPUB1, also provide a 10-word block.

2. Call the routine $STMAP or $STMP1 (set up UNIBUS mapping address) after getting the I/O packet.

3. Call the routine $MPUBM or $MPUB1 (map UNIBUS to memory) before initiating a transfer.

These requirements are detailed in the following three subsections.

Note that these routines are only required when the driver is performing a data transfer.

B.1.1 Allocating a Mapping Register Assignment Block

The status control block (SCB) of an NPR device requires an additional six words. This 6-word mapping register assignment block is located at S.MPR, at the end of the SCB. It does not have to be initialized. Any initial contents are simply overwritten.

The following example shows the allocation of a mapping register assignment block. The code is conditional on the result of an AND operation on the two symbols M$$EXT and M$$MGE (representing extended memory support and memory management unit support, respectively).

```
.IF DF  M$$EXT&M$$MGE
.BLKW   6        ;UMR WORK AREA
.ENDC
```

If the driver does not support parallel NPR operations requiring UMR mapping, it calls $STMAP and $MPUBM. If the driver supports parallel NPR operations requiring UMR mapping, it must call $STMP1 and $MPUB1. In the latter situation, the six additional words starting at S.MPR in the SCB are not used but must still be present. In addition, the driver must provide a 10-word mapping register assignment block for each data transfer to be mapped, as specified in the description of $STMP1 in Chapter 5.

B.1.2 Calling $STMAP or $STMP1

In the coding at the initiator entry point, after the call to $GTPKT, the NPR device driver must call the routine $STMAP or $STMP1. These routines dynamically allocate required UMRs. If UMRs are not available immediately, the driver is blocked. Such blocking, if it occurs, is completely transparent to the driver. The driver resumes processing at fork level when the UMRs have been allocated. The register returns are absolutely identical whether or not blocking has occurred.

$STMAP or $STMP1 stores into U.BUF and U.BUF+2 (in the UCB) a UNIBUS address that causes the appropriate UMR to be selected for mapping the transfer. The call to $STMAP or $STMP1 must be conditional on M$$EXT and M$$MGE.

Because $STMAP and $STMP1 push the address of routine $DQUMR+2 onto the stack before returning to the caller, the driver should not use the stack for temporary data storage when it calls $STMAP or $STMP1.

B.1.3 Calling $MPUBM or $MPUB1

Before executing the transfer, the driver must call $MPUBM or $MPUB1. These routines get the buffer's 22-bit physical address, and load the UNIBUS mapping registers so that transfers are mapped directly to the task's space. The call to $MPUBM or $MPUB1 must be conditional on M$$EXT and M$$MGE.

If the driver calls $STMAP and $MPUBM, the UMRs allocated to it are deallocated during the call to $IODON or $IOALT. If the driver calls $STMP1 and $MPUB1, it must call $DEUMR to deallocate any allocated UMRs before calling $IODON or $IOALT.

B.2 CALLING $ASUMR AND $DEUMR

Some drivers may not require UMRs to be allocated all of the time, and yet require UMRs for periods of time longer than the normal time frame between $GTPKT and $IODON (or $IOALT). In such cases, there is a second procedure for allocating UMRs.

By using the Executive routines $ASUMR and $DEUMR, a driver can dynamically allocate, retain over a desired time frame, and deallocate UMRs. Refer to Section 5.3 for a description of the $ASUMR and $DEUMR routines.

Similar to the $STMAP/$MPUBM procedure, using $ASUMR and $DEUMR also requires the allocation of a 6-word mapping register assignment block. In this instance, however, the block must not be located at offset S.MPR in the SCB. $IODON or $IOALT, when called, will attempt to deallocate the UMRs of a block found at location S.MPR. To avoid this, the mapping register assignment block could, for convenience, be located at S.MPR+2. Alternatively, it could be dynamically allocated from the pool. Figure B-1 details the format of the 6-word block.

M.LNK	Link Word	
M.UMRA	Address of first assigned UMR	
M.UMRN	Number of assigned UMRs *4	
M.UMVL	Low 16 bits mapped by first assigned UMR	
M.UMVH M.BFVH	High 6 bits of physical buffer address	High 2 bits mapped by UMR (in bits 4 and 5)
M.BFVL	Low 16 bits of physical buffer address	

ZK-226-81

Figure B-1 Mapping Register Assignment Block

B.3 STATICALLY ALLOCATING UMRS DURING SYSTEM GENERATION

You can statically assign UMRs during system generation. For systems with extended memory support and memory management unit support, the system generation procedure defines the symbol N$$UMR equal to a fixed number of UMRs, multiplied by 4, that are statically assigned to the system. Before assembling the Executive, you can cause the static allocation of an additional number of UMRs by editing file RSXMC.MAC. The value of the symbol N$$UMR can then be increased to represent the additional number of desired UMRs multiplied by 4.

RSXMC.MAC further defines the following three symbols, which describe the first UMR statically allocated during system generation:

U$$MRN is the I/O page address of the first UMR register available for allocation to the user.

U$$MLO represents the low-order 16 bits of the 18-bit UNIBUS address mapped by this UMR.

U$$MHI represents the high-order two bits of the 18-bit UNIBUS address. These two bits are in bit positions 4 and 5.

These three symbols are not used by the system itself. They are available for the user's information.

APPENDIX C

SYSTEM DATA STRUCTURES AND SYMBOLIC OFFSETS

This appendix describes the system data structures listed in Table C-1.

The data structures are defined by macros in the Executive macro library. To reference any of the data structure offsets from your code, include the macro name in an .MCALL directive and invoke the macro. For example:

```
.MCALL  DCBDF$
DCBDF$                  ;Define DCB offsets
```

NOTE

> All physical offsets and bit definitions are subject to change in future releases of the operating system. Code that accesses system data structures should always use the symbolic offsets rather than the physical offsets.

The first two arguments, <:> and <=>, make all definitions global. If they are left blank, the definitions will be local. The SYSDEF argument causes the variable part of a data structure to be defined.

All of these macros are in the Executive macro library LB:[1,1]EXEMC.MLB. All except ITBDF$ and MTADF$ are also in the Executive definition library LB:[1,1]EXELIB.OLB.

Table C-1
Summary of System Data Structure Macros

Macro	Arguments	Data Structures
ABODF$	<:>,<=>	Task abort and termination notification message codes
CLKDF$	<:>,<=>	Clock queue control block
DCBDF$	<:>,<=>	Device Control Block
EPKDF$	<:>,<=>	Error message block
F11DF$	<:>,<=>,SYSDEF	Files-11 data structures (volume control block, mount list entry, file control block, file window block, locked block list node)
HDRDF$	<:>,<=>	Task header and window block
HWDDF$	<:>,<=>	Hardware register addresses and feature mask definitions
ITBDF$	<:>,<=>,SYSDEF	Interrupt transfer block
LCBDF$	<:>,<=>	Logical assignment control block
MTADF$	<:>,<=>	ANSI magtape data structures (volume set control block)
PCBDF$	<:>,<=>,SYSDEF	Partition control block and attachment descriptor
PKTDF$	<:>,<=>	I/O packet, AST control block, offspring control block, group global event flag control block, and CLI parser block
SCBDF$	<:>,<=>,SYSDEF	Task Control Block
UCBDF$	<:>,<=>,TTDEF,SYSDEF	Unit Control Block

```
        ABODF$

;
; TASK ABORT CODES
;
; NOTE: S.COAD-S.CFLT ARE ALSO SST VECTOR OFFSETS
;
S.CACT=-4.              ;TASK STILL ACTIVE
S.CEXT=-2.              ;TASK EXITTED NORMALLY
S.COAD=0.               ;ODD ADDRESS AND TRAPS TO 4
S.CSGF=2.               ;SEGMENT FAULT
S.CBPT=4.               ;BREAK POINT OR TRACE TRAP
S.CIOT=6.               ;IOT INSTRUCTION
S.CILI=8.               ;ILLEGAL OR RESERVED INSTRUCTION
S.CEMT=10.              ;NON RSX EMT INSTRUCTION
S.CTRP=12.              ;TRAP INSTRUCTION
S.CFLT=14.              ;11/40 FLOATING POINT EXCEPTION
S.CSST=16.              ;SST ABORT-BAD STACK
S.CAST=18.              ;AST ABORT-BAD STACK
S.CABO=20.              ;ABORT VIA DIRECTIVE
S.CLRF=22.              ;TASK LOAD REQUEST FAILURE
S.CCRF=24.              ;TASK CHECKPOINT READ FAILURE
S.IOMG=26.              ;TASK EXIT WITH OUTSTANDING I/O
S.PRTY=28.              ;TASK MEMORY PARITY ERROR
S.CPMD=30.              ;TASK ABORTED WITH PMD REQUEST
S.CINS=32.              ;TASK INSTALLED IN TWO SYSTEMS


;
; TASK TERMINATION NOTIFICATION MESSAGE CODES
;
T.NDNR=0                ;DEVICE NOT READY
T.NDSE=2                ;DEVICE SELECT ERROR
T.NCWF=4                ;CHECKPOINT WRITE FAILURE
T.NCRE=6                ;CARD READER HARDWARE ERROR
T.NDMO=8.               ;DISMOUNT COMPLETE
T.NUER=10.              ;UNRECOVERABLE ERROR
T.NLDN=12.              ;LINK DOWN (NETWORKS)
T.NLUP=14.              ;LINK UP (NETWORKS)
T.NCFI=16.              ;CHECKPOINT FILE INACTIVE
T.NUDE=18.              ;UNRECOVERABLE DEVICE ERROR
T.NMPE=20.              ;MEMORY PARITY ERROR
T.NKLF=22.              ;UCODE LOADER NOT INSTALLED
T.NDEB=24.              ;TASK HAS NO DEBUGGING AID
T.NRCT=26.              ;REPLACEMENT CONTROL TASK NOT INSTALLED
T.NWBL=28.              ;WRITE BACK CACHING DATA LOST
                        ;UNIT WRITE LOCKED
```

CLKDF$

```
                CLKDF$

        ;
        ; CLOCK QUEUE CONTROL BLOCK OFFSET DEFINITIONS
        ;
        ; CLOCK QUEUE CONTROL BLOCK
        ;
        ; THERE ARE SIX TYPES OF CLOCK QUEUE CONTROL BLOCKS. EACH CONTROL
        ; BLOCK HAS THE SAME FORMAT IN THE FIRST FIVE WORDS AND DIFFERS IN
        ; THE REMAINING THREE.
        ;
        ; THE FOLLOWING CONTROL BLOCK TYPES ARE DEFINED:
        ;
        C.MRKT=0                ;MARK TIME REQUEST
        C.SCHD=2                ;TASK REQUEST WITH PERIODIC RESCHEDULING
        C.SSHT=4                ;SINGLE SHOT TASK REQUEST
        C.SYST=6                ;SINGLE SHOT INTERNAL SYSTEM SUBROUTINE (IDENT)
        C.SYTK=8.               ;SINGLE SHOT INTERNAL SYSTEM SUBROUTINE (TASK)
        C.CSTP=10.              ;CLEAR STOP BIT (CONDITIONALIZED ON SHUFFLING)


        ;
        ; CLOCK QUEUE CONTROL BLOCK TYPE INDEPENDENT OFFSET DEFINITIONS
        ;
                .ASECT
        .=0
000000  C.LNK:  .BLKW   1       ;CLOCK QUEUE THREAD WORD
000002  C.RQT:  .BLKB   1       ;REQUEST TYPE
000003  C.EFN:  .BLKB   1       ;EVENT FLAG NUMBER (MARK TIME ONLY)
000004  C.TCB:  .BLKW   1       ;TCB ADDR OR SYSTEM SUBROUTINE IDENTIFICATION
000006  C.TIM:  .BLKW   2       ;ABSOLUTE TIME WHEN REQUEST COMES DUE


        ;
        ; CLOCK QUEUE CONTROL BLOCK-MARK TIME DEPENDENT OFFSET DEFINITIONS
        ;
        .=C.TIM+4               ;START OF DEPENDENT AREA
000012  C.AST:  .BLKW   1       ;AST ADDRESS
000014  C.SRC:  .BLKW   1       ;FLAG MASK WORD FOR 'BIS' SOURCE
000016  C.DST:  .BLKW   1       ;ADDRESS OF 'BIS' DESTINATION


        ;
        ; CLOCK QUEUE CONTROL BLOCK-PERIODIC RESCHEDULING DEPENDENT OFFSET
        ; DEFINITIONS
        ;
        .=C.TIM+4               ;START OF DEPENDENT AREA
000012  C.RSI:  .BLKW   2       ;RESCHEDULE INTERVAL IN CLOCK TICKS
000016  C.UIC:  .BLKW   1       ;SCHEDULING UIC


        ;
        ; CLOCK QUEUE CONTROL BLOCK-SINGLE SHOT DEPENDENT OFFSET DEFINITIONS
        ;
        .=C.TIM+4               ;START OF DEPENDENT AREA
000012          .BLKW   2       ;TWO UNUSED WORDS
000016          .BLKW   1       ;SCHEDULING UIC
```

```
        ;
        ; CLOCK QUEUE CONTROL BLOCK-SINGLE SHOT INTERNAL SUBROUTINE OFFSET
        ; DEFINITIONS
        ;
        ; THERE ARE TWO TYPE CODES FOR THIS TYPE OF REQUEST:
        ;
        ;       TYPE 6 = SINGLE SHOT INTERNAL SUBROUTINE WITH A 16 BIT VALUE
        ;                AS AN IDENTIFIER.
        ;
        ;       TYPE 8 = SINGLE SHOT INTERNAL SUBROUTINE WITH A TCB ADDRESS
        ;                AS AN IDENTIFIER.
        ;
        .=C.TIM+4                       ;START OF DEPENDENT AREA
000012  C.SUB:  .BLKW   1               ;SUBROUTINE ADDRESS
000014  C.AR5:  .BLKW   1               ;RELOCATION BASE (FOR LOADABLE DRIVERS)
000016          .BLKW   1               ;ONE UNUSED WORD

000020  C.LGTH=.                        ;LENGTH OF CLOCK QUEUE CONTROL BLOCK

                .PSECT
```

DCBDF$

```
                DCBDF$

        ;
        ; DEVICE CONTROL BLOCK
        ;
        ; THE DEVICE CONTROL BLOCK (DCB) DEFINES GENERIC INFORMATION ABOUT
        ; A DEVICE TYPE AND THE LOWEST AND HIGHEST UNIT NUMBERS. THERE IS
        ; AT LEAST ONE DCB FOR EACH DEVICE TYPE IN A SYSTEM. FOR EXAMPLE,
        ; IF THERE ARE TELETYPES IN A SYSTEM, THEN THERE IS AT LEAST ONE
        ; DCB WITH THE DEVICE NAME 'TT'. IF PART OF THE TELETYPES WERE
        ; INTERFACED VIA DL11-A'S AND THE REST VIA A DH11, THEN THERE
        ; WOULD BE TWO DCB'S. ONE FOR ALL DL11-A INTERFACED TELETYPES,
        ; AND ONE FOR ALL DH11 INTERFACED TELETYPES.
        ;
                .ASECT
        .=0
000000  D.LNK:  .BLKW   1       ;LINK TO NEXT DCB
000002  D.UCB:  .BLKW   1       ;POINTER TO FIRST UNIT CONTROL BLOCK
000004  D.NAM:  .BLKW   1       ;GENERIC DEVICE NAME
000006  D.UNIT: .BLKB   1       ;LOWEST UNIT NUMBER COVERED BY THIS DCB
000007          .BLKB   1       ;HIGHEST UNIT NUMBER COVERED BY THIS DCB
000010  D.UCBL: .BLKW   1       ;LENGTH OF EACH UNIT CONTROL BLOCK IN BYTES
000012  D.DSP:  .BLKW   1       ;POINTER TO DRIVER DISPATCH TABLE
000014  D.MSK:  .BLKW   1       ;LEGAL FUNCTION MASK CODES 0-15.
000016          .BLKW   1       ;CONTROL FUNCTION MASK CODES 0-15.
000020          .BLKW   1       ;NOP'ED FUNCTION MASK CODES 0-15.
000022          .BLKW   1       ;ACP FUNCTION MASK CODES 0-15.
000024          .BLKW   1       ;LEGAL FUNCTION MASK CODES 16.-31.
000026          .BLKW   1       ;CONTROL FUNCTION MASK CODES 16.-31.
000030          .BLKW   1       ;NOP'ED FUNCTION MASK CODES 16.-31.
000032          .BLKW   1       ;ACP FUNCTION MASK CODES 16.-31.
000034  D.PCB:  .BLKW   1       ;LOADABLE DRIVER PCB ADDRESS

                .PSECT


        ;
        ; DRIVER DISPATCH TABLE OFFSET DEFINITIONS
        ;
        D.VDEB=177776           ;DEALLOCATE INTERNAL BUFFERS (FD TTDRV)
        D.VINI=0                ;DEVICE INITIATOR
        D.VCAN=2                ;CANCEL CURRENT I/O FUNCTION
        D.VOUT=4                ;DEVICE TIMEOUT
        D.VPWF=6                ;POWERFAIL RECOVERY
```

EPKDF$

```
                EPKDF$

        ;
        ; ERROR MESSAGE BLOCK DEFINITIONS
        ;
                .ASECT


        ;
        ; HEADER SUBPACKET
        ;
        ;
        ;       +-----------------------------------------------+
        ;       | SUBPACKET LENGTH IN BYTES                     |
        ;       +-----------------------------------------------+
        ;       | SUBPACKET FLAGS                               |
        ;       +-----------------------+-----------------------+
        ;       | FORMAT IDENTIFICATION | OPERATING SYSTEM CODE |
        ;       +-----------------------+-----------------------+
        ;       | OPERATING SYSTEM IDENTIFICATION               |
        ;       |                                               |
        ;       +-----------------------+-----------------------+
        ;       | FLAGS                 | CONTEXT CODE          |
        ;       +-----------------------+-----------------------+
        ;       | ENTRY SEQUENCE                                |
        ;       +-----------------------------------------------+
        ;       | ERROR SEQUENCE                                |
        ;       +-----------------------+-----------------------+
        ;       | ENTRY TYPE SUBCODE    | ENTRY TYPE CODE       |
        ;       +-----------------------+-----------------------+
        ;       | TIME STAMP                                    |
        ;       |                                               |
        ;       |                                               |
        ;       +-----------------------+-----------------------+
        ;       | RESERVED              | PROCESSOR TYPE        |
        ;       +-----------------------+-----------------------+
        ;       | PROCESSOR IDENTIFICATION (URM)                |
        ;       +-----------------------------------------------+
        ;
        ;
        .=0
000000  E$HLGH: .BLKW   1       ; SUBPACKET LENGTH IN BYTES
000002  E$HSBF: .BLKW   1       ; SUBPACKET FLAGS
000004  E$HSYS: .BLKB   1       ; OPERATING SYSTEM CODE
000005  E$HIDN: .BLKB   1       ; FORMAT IDENTIFICATION
000006  E$HSID: .BLKB   4       ; OPERATING SYSTEM IDENTIFICATION
000012  E$HCTX: .BLKB   1       ; CONTEXT CODE
000013  E$HFLG: .BLKB   1       ; FLAGS
000014  E$HENS: .BLKW   1       ; ENTRY SEQUENCE NUMBER
000016  E$HERS: .BLKW   1       ; ERROR SEQUENCE NUMBER
000020  E$HENC:                 ; ENTRY CODE
000020  E$HTYC: .BLKB   1       ; ENTRY TYPE CODE
000021  E$HTYS: .BLKB   1       ; ENTRY TYPE SUBCODE
000022  E$HTIM: .BLKB   6       ; TIME STAMP
000030  E$HPTY: .BLKB   1       ; PROCESSOR TYPE
000031          .BLKB   1       ; RESERVED
000032  E$HURM: .BLKW   1       ; PROCESSOR IDENTIFICATION (URM)
                .EVEN
000034  E$HLEN:                 ; LENGTH
```

```
;
; SUBPACKET FLAGS FOR E$HSBF
;
        SM.ERR  =      1 ; ERROR PACKET
        SM.HDR  =      1 ; HEADER SUBPACKET
        SM.TSK  =      2 ; TASK SUBPACKET
        SM.DID  =      4 ; DEVICE IDENTIFICATION SUBPACKET
        SM.DOP  =     10 ; DEVICE OPERATION SUBPACKET
        SM.DAC  =     20 ; DEVICE ACTIVITY SUBPACKET
        SM.DAT  =     40 ; DATA SUBPACKET
        SM.MBC  = 20000  ; 22-BIT MASSBUS CONTROLLER PRESENT
        SM.CMD  = 40000  ; ERROR LOG COMMAND PACKET
        SM.ZER  =100000  ; ZERO I/O COUNTS

;
; CODES FOR FIELD E$HIDN
;
        EH$FOR  =      1 ; CURRENT PACKET FORMAT

;
; FLAGS FOR THE ERROR LOG FLAGS BYTE ($ERFLA) IN THE EXEC
;
        ES.INI  =      1 ; ERROR LOG INITIALIZED
        ES.DAT  =      2 ; ERROR LOG RECEIVING DATA PACKETS
        ES.LIM  =      4 ; ERROR LIMITING ENABLED
        ES.LOG  =     10 ; ERROR LOGGING ENABLED

;
; TYPE AND SUBTYPE CODES FOR FIELDS E$HTYC AND E$HTYS
;
;       SYMBOLS WITH NAMES E$CXXX ARE TYPE CODES FOR FIELD E$HTYC,
;       SYMBOLS WITH NAMES E$SXXX ARE SUBTYPE CODES FOR FIELD E$HTYS.
;
        E$CCMD  =      1 ; ERROR LOG CONTROL
        E$SSTA  =      1 ;       ERROR LOG STATUS CHANGE
        E$SSWI  =      2 ;       SWITCH LOGGING FILES
        E$SAPP  =      3 ;       APPEND FILE
        E$SBAC  =      4 ;       DECLARE BACKUP FILE
        E$SSHO  =      5 ;       SHOW
        E$SCHL  =      6 ;       CHANGE LIMITS

        E$CERR  =      2 ; DEVICE ERRORS
        E$SDVH  =      1 ;       DEVICE HARD ERROR
        E$SDVS  =      2 ;       DEVICE SOFT ERROR
        E$STMO  =      3 ;       DEVICE INTERRUPT TIMEOUT
        E$SUNS  =      4 ;       DEVICE UNSOLICITED INTERRUPT

        E$CDVI  =      3 ; DEVICE INFORMATION
        E$SDVI  =      1 ;       DEVICE INFORMATION MESSAGE

        E$CDCI  =      4 ; DEVICE CONTROL INFORMATION
        E$SMOU  =      1 ;       DEVICE MOUNT
        E$SDMO  =      2 ;       DEVICE DISMOUNT
        E$SRES  =      3 ;       DEVICE COUNT RESET
        E$SRCT  =      4 ;       BLOCK REPLACEMENT

        E$CCPU  =      5 ; CPU DETECTED ERRORS
        E$SMEM  =      1 ;       MEMORY ERROR
        E$SINT  =      2 ;       UNEXPECTED INTERRUPT

        E$CSYS  =      6 ; SYSTEM CONTROL INFORMATION
        E$SPWR  =      1 ;       POWER RECOVERY
```

```
        E$CCTL  =      7 ; CONTROL INFORMATION
        E$STIM  =      1 ;       TIME CHANGE
        E$SCRS  =      2 ;       SYSTEM CRASH
        E$SLOA  =      3 ;       DEVICE DRIVER LOAD
        E$SUNL  =      4 ;       DEVICE DRIVER UNLOAD
        E$SHRC  =      5 ;       RECONFIGURATION STATUS CHANGE
        E$SMES  =      6 ;       MESSAGE

        E$CSDE  =     10 ; SOFTWARE DETECTED EVENTS
        E$SABO  =      1 ;       TASK ABORT

;
; CODES FOR CONTEXT CODE ENTRY E$HCTX
;
        EH$NOR  =      1 ; NORMAL ENTRY
        EH$STA  =      2 ; START ENTRY
        EH$CRS  =      3 ; CRASH ENTRY

;
; CODES FOR FLAGS ENTRY E$HFLG
;
        EH$VIR  =      1 ; ADDRESSES ARE VIRTUAL
        EH$EXT  =      2 ; ADDRESSES ARE EXTENDED
        EH$COU  =      4 ; ERROR COUNTS SUPPLIED


;
; TASK SUBPACKET
;
;       +-----------------------------------------------+
;       | TASK SUBPACKET LENGTH                         |
;       +-----------------------------------------------+
;       | TASK NAME IN RAD50                            |
;       |                                               |
;       +-----------------------------------------------+
;       | TASK UIC                                      |
;       +-----------------------------------------------+
;       | TASK TI: DEVICE NAME                          |
;       +-----------------------+-----------------------+
;       | FLAGS                 | TASK TI: UNIT NUMBER  |
;       +-----------------------+-----------------------+
;
.=0
000000  E$TLGH: .BLKW   1       ; TASK SUBPACKET LENGTH
000002  E$TTSK: .BLKW   2       ; TASK NAME IN RAD50
000006  E$TUIC: .BLKW   1       ; TASK UIC
000010  E$TTID: .BLKB   2       ; TASK TI: DEVICE NAME
000012  E$TTIU: .BLKB   1       ; TASK TI: UNIT
000013  E$TFLG: .BLKB   1       ; FLAGS
                .EVEN
000014  E$TLEN:

;
; FLAGS FOR ENTRY E$TFLG
;
        ET$PRV  =      1 ; TASK IS PRIVILEGED
        ET$PRI  =      2 ; TERMINAL IS PRIVILEGED
```

```
         ;
         ; DEVICE IDENTIFICATION SUBPACKET
         ;
         ;       +----------------------------------------------+
         ;       | DEVICE IDENTIFICATION SUBPACKET LENGTH       |
         ;       +----------------------------------------------+
         ;       | DEVICE MNEMONIC NAME                         |
         ;       +-----------------------+----------------------+
         ;       | CONTROLLER NUMBER     | DEVICE UNIT NUMBER   |
         ;       +-----------------------+----------------------+
         ;       | PHYSICAL SUBUNIT #    | PHYSICAL UNIT #      |
         ;       +-----------------------+----------------------+
         ;       | PHYSICAL DEVICE MNEMONIC (RSX-11M-PLUS ONLY) |
         ;       +-----------------------+----------------------+
         ;       | RESERVED              | FLAGS                |
         ;       +-----------------------+----------------------+
         ;       | VOLUME NAME OF MOUNTED VOLUME                |
         ;       |                                              |
         ;       |                                              |
         ;       |                                              |
         ;       |                                              |
         ;       |                                              |
         ;       +----------------------------------------------+
         ;       | PACK IDENTIFICATION                          |
         ;       |                                              |
         ;       +----------------------------------------------+
         ;       | DEVICE TYPE CLASS                            |
         ;       +----------------------------------------------+
         ;       | DEVICE TYPE                                  |
         ;       |                                              |
         ;       +----------------------------------------------+
         ;       | I/O OPERATION COUNT LONGWORD                 |
         ;       |                                              |
         ;       +-----------------------+----------------------+
         ;       | HARD ERROR COUNT      | SOFT ERROR COUNT     |
         ;       +-----------------------+----------------------+
         ;       | BLOCKS TRANSFERRED COUNT (RSX-11M-PLUS ONLY) |
         ;       |                                              |
         ;       +----------------------------------------------+
         ;       | CYLINDERS CROSSED COUNT (RSX-11M-PLUS ONLY)  |
         ;       |                                              |
         ;       +----------------------------------------------+
         ;
         .=0
000000   E$ILGH: .BLKW   1        ; DEVICE IDENTIFICATION SUBPACKET LENGTH
000002   E$ILDV: .BLKW   1        ; DEVICE MNEMONIC NAME
000004   E$ILUN: .BLKB   1        ; DEVICE UNIT NUMBER
000005   E$IPCO: .BLKB   1        ; CONTROLLER NUMBER
000006   E$IPUN: .BLKB   1        ; PHYSICAL UNIT NUMBER
000007   E$IPSU: .BLKB   1        ; PHYSICAL SUBUNIT NUMBER

                 .IF DF R$$MPL

         E$IPDV: .BLKW   1        ; PHYSICAL DEVICE MNEMONIC

                 .ENDC ; R$$MPL
```

```
000010  E$IFLG: .BLKB   1       ; FLAGS
000011          .BLKB   1       ; RESERVED
000012  E$IVOL: .BLKB   12.     ; VOLUME NAME
000026  E$IPAK: .BLKB   4       ; PACK IDENTIFICATION
000032  E$IDEV:                 ; DEVICE TYPE
000032  E$IDCL: .BLKW   1       ; DEVICE TYPE CLASS
000034  E$IDTY: .BLKW   2       ; DEVICE TYPE
000040  E$IOPR: .BLKW   2       ; I/O OPERATION COUNT LONGWORD
000044  E$IERS: .BLKB   1       ; SOFT ERROR COUNT
000045  E$IERH: .BLKB   1       ; HARD ERROR COUNT

                .IF DF R$$MPL

        E$IBLK: .BLKW   2       ; BLOCKS TRANSFERRED COUNT
        E$ICYL: .BLKW   2       ; CYLINDERS CROSSED COUNT

                .ENDC ; R$$MPL

                .EVEN
000046  E$ILEN:                 ; SUBPACKET LENGTH

        ;
        ; FLAGS FOR FIELD E$IFLG
        ;
                EI$SUB  =     1 ; SUBCONTROLLER DEVICE


        ;
        ; DEVICE OPERATION SUBPACKET
        ;
        ;       +------------------------------------------------+
        ;       | DEVICE OPERATION SUBPACKET LENGTH              |
        ;       +------------------------------------------------+
        ;       | TASK NAME IN RAD50                             |
        ;       |                                                |
        ;       +------------------------------------------------+
        ;       | TASK UIC                                       |
        ;       +------------------------------------------------+
        ;       | TASK TI: LOGICAL DEVICE MNEMONIC               |
        ;       +------------------------+-----------------------+
        ;       | RESERVED               | TASK TI: DEVICE UNIT  |
        ;       +------------------------+-----------------------+
        ;       | I/O FUNCTION CODE                              |
        ;       +------------------------+-----------------------+
        ;       | RESERVED               | OPERATION FLAGS       |
        ;       +------------------------+-----------------------+
        ;       | TRANSFER OPERATION ADDRESS                     |
        ;       |                                                |
        ;       +------------------------------------------------+
        ;       | TRANSFER OPERATION BYTE COUNT                  |
        ;       +------------------------------------------------+
        ;       | CURRENT OPERATION RETRY COUNT                  |
        ;       +------------------------------------------------+
        ;
        .=0
000000  E$OLGN: .BLKW   1       ; SUBPACKET LENGTH
000002  E$OTSK: .BLKW   2       ; TASK NAME IN RAD50
000006  E$OUIC: .BLKW   1       ; TASK UIC
000010  E$OTID: .BLKB   2       ; TASK TI: LOGICAL DEVICE MNEMONIC
000012  E$OTIU: .BLKB   1       ; TASK TI: LOGICAL DEVICE UNIT
000013          .BLKB   1       ; RESERVED
000014  E$OFNC: .BLKW   1       ; I/O FUNCTION CODE
000016  E$OFLG: .BLKB   1       ; OPERATION FLAGS
000017          .BLKB   1       ; RESERVED
```

```
000020  E$OADD: .BLKW   2        ; TRANSFER OPERATION ADDRESS
000024  E$OSIZ: .BLKW   1        ; TRANSFER OPERATION BYTE COUNT
000026  E$ORTY: .BLKW   1        ; CURRENT OPERATION RETRY COUNT
                .EVEN
000030  E$OLEN:                  ; DEVICE OPERATION SUBPACKET LENGTH

        ;
        ; FLAGS FOR FIELD E$OFLG
        ;
                EO$TRA  =     1 ; TRANSFER OPERATION
                EO$DMA  =     2 ; DMA DEVICE
                EO$EXT  =     4 ; EXTENDED ADDRESSING DEVICE
                EO$PIP  =    10 ; DEVICE IS POSITIONING


        ;
        ; I/O ACTIVITY SUBPACKET
        ;
        ;       +------------------------------------------------+
        ;       | I/O ACTIVITY SUBPACKET LENGTH                  |
        ;       +------------------------------------------------+
        ;
        .=0
000000  E$ALGH: .BLKW   1        ; SUBPACKET LENGTH


        ;
        ; I/O ACTIVITY SUBPACKET ENTRY
        ;
        ;       +------------------------------------------------+
        ;       | LOGICAL DEVICE NAME MNEMONIC                   |
        ;       +-----------------------+------------------------+
        ;       | CONTROLLER NUMBER     | LOGICAL DEVICE UNIT    |
        ;       +-----------------------+------------------------+
        ;       | PHYSICAL SUBUNIT #    | PHYSICAL UNIT NUMBER   |
        ;       +-----------------------+------------------------+
        ;       | PHYSICAL DEVICE MNEMONIC (RSX-11M-PLUS ONLY)   |
        ;       +-----------------------+------------------------+
        ;       | TASK TI: LOGICAL UNIT | DEVICE FLAGS           |
        ;       +-----------------------+------------------------+
        ;       | REQUESTING TASK NAME IN RAD50                  |
        ;       |                                                |
        ;       +------------------------------------------------+
        ;       | REQUESTING TASK UIC                            |
        ;       +------------------------------------------------+
        ;       | TASK TI: LOGICAL DEVICE NAME                   |
        ;       +------------------------------------------------+
        ;       | I/O FUNCTION CODE                              |
        ;       +-----------------------+------------------------+
        ;       | RESERVED              | FLAGS                  |
        ;       +-----------------------+------------------------+
        ;       | TRANSFER OPERATION ADDRESS                     |
        ;       |                                                |
        ;       +------------------------------------------------+
        ;       | TRANSFER OPERATION BYTE COUNT                  |
        ;       +------------------------------------------------+
        ;
```

```
                .=0
000000  E$ALDV: .BLKW   1       ; LOGICAL DEVICE NAME MNEMONIC
000002  E$ALUN: .BLKB   1       ; LOGICAL DEVICE UNIT
000003  E$APCO: .BLKB   1       ; CONTROLLER NUMBER
000004  E$APUN: .BLKB   1       ; PHYSICAL UNIT NUMBER
000005  E$APSU: .BLKB   1       ; PHYSICAL SUBUNIT NUMBER

                .IF DF R$$MPL

        E$APDV: .BLKW   1       ; PHYSICAL DEVICE MNEMONIC

                .ENDC ; R$$MPL

000006  E$ADFG: .BLKB   1       ; DEVICE FLAGS
000007  E$ATIU: .BLKB   1       ; TASK TI: LOGICAL UNIT
000010  E$ATSK: .BLKW   2       ; REQUESTING TASK NAME IN RAD50
000014  E$AUIC: .BLKW   1       ; REQUESTING TASK UIC
000016  E$ATID: .BLKW   1       ; TASK TI: LOGICAL DEVICE NAME
000020  E$AFNC: .BLKW   1       ; I/O FUNCTION CODE
000022  E$AFLG: .BLKB   1       ; FLAGS
000023          .BLKB   1       ; RESERVED
000024  E$AADD: .BLKW   2       ; TRANSFER OPERATION ADDRESS
000030  E$ASIZ: .BLKW   1       ; TRANSFER OPERATION BYTE COUNT
                .EVEN
000032  E$ALEN:                 ; SUBPACKET ENTRY LENGTH

        ;
        ; FLAGS FOR FIELD E$ADFG
        ;
                EA$SUB  =       1 ; SUBCONTROLLER DEVICE

        ;
        ; FLAGS FOR FIELD E$AFLG
        ;
                EA$TRA  =       1 ; TRANSFER OPERATION
                EA$DMA  =       2 ; DMA DEVICE
                EA$EXT  =       4 ; DEVICE HAS EXTENDED ADDRESSING
                EA$PIP  =      10 ; DEVICE IS POSITIONING

                .PSECT
```

F11DF$

```
                F11DF$  ,,SYSDEF

        ;
        ; VOLUME CONTROL BLOCK
        ;
                .ASECT
        .=0
000000  V.TRCT: .BLKW   1       ; TRANSACTION COUNT

                .IF DF R$$11M

000002  V.TYPE: .BLKB   1       ; VOLUME TYPE DESCRIPTOR
        VT.SL1= 1               ; FILES-11 STRUCTURE LEVEL 1
        VT.ANS= 10              ; ANSI LABELED TAPE
        VT.UNL= 11              ; UNLABELED TAPE
000003  V.VCHA: .BLKB   1       ; VOLUME CHARACTERISTICS
        VC.SLK= 1               ; CLEAR VOLUME VALID ON DISMOUNT
        VC.HLK= 2               ; UNLOAD THE VOLUME ON DISMOUNT
        VC.DEA= 4               ; DEALLOCATE THE VOLUME ON DISMOUNT
        VC.PUB= 10              ; SET (CLEAR) US.PUB ON DISMOUNT
000004  V.LABL: .BLKB   14      ; VOLUME LABEL (ASCII)
000020  V.PKSR: .BLKW   2       ; PACK SERIAL NUMBER FOR ERROR LOGGING

000024  V.SLEN:                 ; LENGTH OF SHORT VCB

                .ENDC ;R$$11M

000024  V.IFWI: .BLKW   1       ; INDEX FILE WINDOW

                .IF DF R$$11D

        V.STD:  .BLKW   1       ; STD OF TASK CHARGED WITH NODE

                .ENDC ;R$$11D

000026  V.FCB:  .BLKW   2       ; FILE CONTROL BLOCK LIST HEAD
000032  V.IBLB: .BLKB   1       ; INDEX BIT MAP 1ST LBN HIGH BYTE
000033  V.IBSZ: .BLKB   1       ; INDEX BIT MAP SIZE IN BLOCKS
000034          .BLKW   1       ; INDEX BITMAP 1ST LBN LOW BITS
000036  V.FMAX: .BLKW   1       ; MAX NO. OF FILES ON VOLUME
000040  V.WISZ: .BLKB   1       ; DEFAULT SIZE OF WINDOW IN RTRV PTRS
                                ; VALUE IS < 128.
000041  V.SBCL: .BLKB   1       ; STORAGE BIT MAP CLUSTER FACTOR
000042  V.SBSZ: .BLKW   1       ; STORAGE BIT MAP SIZE IN BLOCKS
000044  V.SBLB: .BLKB   1       ; STORAGE BIT MAP 1ST LBN HIGH BYTE
000045  V.FIEX: .BLKB   1       ; DEFAULT FILE EXTEND SIZE
000046          .BLKW   1       ; STORAGE BIT MAP 1ST LBN LOW BITS

                .IF DF R$$11M

000050  V.VOWN: .BLKW   1       ; VOLUME OWNER'S UIC
000052  V.VPRO: .BLKW   1       ; VOLUME PROTECTION

                .ENDC ;R$$11M

000054  V.FPRO: .BLKW   1       ; VOLUME DEFAULT FILE PROTECTION
000056  V.FRBK: .BLKB   1       ; NUMBER OF FREE BLOCKS ON VOLUME HIGH BYTE
000057  V.LRUC: .BLKB   1       ; COUNT OF AVAILABLE LRU SLOTS IN FCB LIST
000060          .BLKW   1       ; NUMBER OF FREE BLOCKS ON VOLUME LOW BITS
000062  V.STS:  .BLKB   1       ; VOLUME STATUS BYTE, CONTAINING THE FOLLOWING
        VS.IFW= 1               ; INDEX FILE IS WRITE ACCESSED
        VS.BMW= 2               ; STORAGE BITMAP FILE IS WRITE ACCESSED
```

```
000063  V.FFNU: .BLKB   1       ; FIRST FREE INDEX FILE BITMAP BLOCK
000064  V.EXT:  .BLKW   1       ; POINTER TO VCB EXTENSION

000066  V.LGTH:                 ; SIZE IN BYTES OF VCB


        ;
        ; MOUNT LIST ENTRY
        ;
        ; EACH ENTRY ALLOWS ACCESS TO A SPECIFIED USER FOR A NON-PUBLIC DEVICE
        ;
        ; TO ALLOW EXPANSION, ONLY THE ONLY TYPE CODE DEFINED IS "1" FOR
        ; DEVICE ACCESS BLOCKS
        ;
                .ASECT
        .=0
000000  M.LNK:  .BLKW   1       ; LINK WORD
000002  M.TYPE: .BLKB   1       ; TYPE OF ENTRY
        MT.MLS= 1               ; MOUNTED VOLUME USER ACCESS LIST
000003  M.ACC:  .BLKB   1       ; NUMBER OF ACCESSES
000004  M.DEV:  .BLKW   1       ; DEVICE UCB
000006  M.TI:   .BLKW   1       ; ACCESSOR TI: UCB

000010  M.LEN:                  ; LENGTH OF ENTRY


        ;
        ; FILE CONTROL BLOCK
        ;
                .ASECT
        .=0
000000  F.LINK: .BLKW   1       ; FCB CHAIN POINTER

                .IF DF R$$11D

        F.FEXT: .BLKW   1       ; POINTER TO EXTENSION FCB
        F.STD:  .BLKW   1       ; STD OF TASK CHARGED WITH NODE

                .ENDC ;R$$11D

000002  F.FNUM: .BLKW   1       ; FILE NUMBER
000004  F.FSEQ: .BLKW   1       ; FILE SEQUENCE NUMBER
000006          .BLKB   1       ; NOT USED
000007  F.FSQN: .BLKB   1       ; FILE SEGMENT NUMBER
000010  F.FOWN: .BLKW   1       ; FILE OWNER'S UIC
000012  F.FPRO: .BLKW   1       ; FILE PROTECTION CODE
000014  F.UCHA: .BLKB   1       ; USER CONTROLLED CHARACTERISTICS
000015  F.SCHA: .BLKB   1       ; SYSTEM CONTROLLED CHARACTERISTICS
000016  F.HDLB: .BLKW   2       ; FILE HEADER LOGICAL BLOCK NUMBER

                                ; BEGINNING OF STATISTICS BLOCK
000022  F.LBN:  .BLKW   2       ; LBN OF VIRTUAL BLOCK 1 IF CONTIGUOUS
                                ; 0 IF NON CONTIGUOUS
000026  F.SIZE: .BLKW   2       ; SIZE OF FILE IN BLOCKS
000032  F.NACS: .BLKB   1       ; NO. OF ACCESSES
000033  F.NLCK: .BLKB   1       ; NO. OF LOCKS
```

```
000012  S.STBK=.-F.LBN                ; SIZE OF STATISTICS BLOCK

000034  F.STAT:                       ; FCB STATUS WORD
000034  F.NWAC: .BLKB    1            ; NUMBER OF WRITE ACCESSORS
000035          .BLKB    1            ; STATUS BITS FOR FCB CONSISTING OF
        FC.WAC= 100000                ; SET IF FILE ACCESSED FOR WRITE
        FC.DIR= 40000                 ; SET IF FCB IS IN DIRECTORY LRU
        FC.CEF= 20000                 ; SET IF DIRECTORY EOF NEEDS UPDATING
        FC.FCO= 10000                 ; SET IF TRYING TO FORCE DIRECTORY CONTIG
000036  F.DREF: .BLKW    1            ; DIRECTORY EOF BLOCK NUMBER
000040  F.DRNM: .BLKW    1            ; 1ST WORD OF DIRECTORY NAME

                .IF DF R$$11M

000042  F.FEXT: .BLKW    1            ; POINTER TO EXTENSION FCB

                .ENDC ;R$$11M

000044  F.FVBN: .BLKW    2            ; STARTING VBN OF THIS FILE SEGMENT
000050  F.LKL:  .BLKW    1            ; POINTER TO LOCKED BLOCK LIST FOR FILE
000052  F.WIN:  .BLKW    1            ; WINDOW BLOCK LIST FOR THIS FILE

000054  F.LGTH:                       ; SIZE IN BYTES OF FCB


        ;
        ; WINDOW
        ;
                .ASECT
        .=0
000000  W.ACT:                        ; NUMBER OF ACTIVE MAPPING POINTERS
                                      ;  WHEN NO SECONDARY POOL
000000  W.BLKS:                       ; BLOCK SIZE OF SECONDARY POOL SEGMENT
                                      ;  WHEN SECONDARY POOL
000000  W.CTL:  .BLKW    1            ; LOW BYTE = # OF MAP ENTRIES ACTIVE
                                      ; HIGH BYTE CONSISTS OF CONTROL BITS
        WI.RDV= 400                   ; READ VIRTUAL BLOCK ALLOWED IF SET
        WI.WRV= 1000                  ; WRITE VIRTUAL BLOCK ALLOWED IF SET
        WI.EXT= 2000                  ; EXTEND ALLOWED IF SET
        WI.LCK= 4000                  ; SET IF LOCKED AGAINST SHARED ACCESS
        WI.DLK= 10000                 ; SET IF DEACCESS LOCK ENABLED

                .IF DF R$$11M

        WI.PND= 20000                 ; WINDOW TURN PENDING BIT

                .ENDC ;R$$11M

        WI.EXL= 40000                 ; SET IF MANUAL UNLOCK DESIRED
        WI.WCK= 100000                ; DATA CHECK ALL WRITES TO FILE

                .IF NDF R$$11M        ; IF NOT RSX-11

        W.FCB:  .BLKW    1            ; FILE CONTROL BLOCK ADDRESS
        W.STD:  .BLKW    1            ; STD OF TASK CHARGED WITH WIDOW NODE
        W.VBN:  .BLKB    1            ; HIGH BYTE OF 1ST VBN MAPPED BY WINDOW
        W.WISZ: .BLKB    1            ; SIZE IN RTRV PTRS OF WINDOW (7 BITS)
                .BLKW    1            ; LOW ORDER WORD OF 1ST VBN MAPPED
        W.LKL:  .BLKW    1            ; POINTER TO LIST OF USERS LOCKED BLOCKS
        W.WIN:  .BLKW    1            ; WINDOW BLOCK LIST LINK WORD
        W.RTRV:                       ; OFFSET TO 1ST RETRIEVAL POINTER IN WINDOW
```

```
                .IFF                ; IF RSX-11

000002  W.IOC:  .BLKB    1          ; COUNT OF I/O THROUGH THIS WINDOW
000003          .BLKB    1          ; RESERVED
000004  W.FCB:  .BLKW    1          ; FILE CONTROL BLOCK ADDRESS
000006  W.LKL:  .BLKW    1          ; POINTER TO LIST OF USERS LOCKED BLOCKS
000010  W.WIN:  .BLKW    1          ; WINDOW BLOCK LIST LINK WORD

                .IF NB SYSDEF       ; IF SYSDEF SPECIFIED IN CALL

                .IF NDF P$$WND      ; IF SECONDARY POOL WINDOWS NOT ALLOWED

        ;
        ; NON-SECONDARY POOL WINDOW BLOCK
        ;       IF SECONDARY POOL WINDOWS ARE NOT ENABLED, THE WINDOW BLOCK
        ;       CONTAINS THE CONTROL INFORMATION AND RETRIEVAL POINTERS.
        ;
000012  W.VBN:  .BLKB    1          ; HIGH BYTE OF 1ST VBN MAPPED BY WINDOW
000013  W.MAP:                      ; DEFINE LABEL WITH ODD ADDR TO CATCH BAD REFS
000013  W.WISZ: .BLKB    1          ; SIZE IN RTRV PTRS OF WINDOW (7 BITS)
000014          .BLKW    1          ; LOW ORDER WORD OF 1ST VBN MAPPED
000016  W.RTRV:                     ; OFFSET TO 1ST RETRIEVAL POINTER IN WINDOW

                .IFF                ; IF WINDOWS IN SECONDARY POOL

        ;
        ; SECONDARY POOL WINDOW CONTROL AND MAPPING BLOCK
        ;       IF SECONDARY POOL WINDOW BLOCKS ARE ENABLED, LUTN2 POINTS
        ;       TO A CONTROL BLOCK IN SYSTEM POOL WHICH CONTAINS THE
        ;       FOLLOWING CONTROL FIELDS AND THE MAPPING INFORMATION
        ;       FOR THE SECONDARY POOL WINDOW.
        ;
        W.MAP:  .BLKW    1          ; ADDR TO THE MAPPING PTRS IN SECONDARY POOL

        ;
        ; SECONDARY POOL WINDOW
        ;       IF SECONDARY POOL WINDOW BLOCKS ARE ENABLED, THE RETRIEVAL
        ;       POINTERS ARE MAINTAINED IN SECONDARY POOL IN THE FOLLOWING
        ;       FORMAT.
        ;
        .=0
                ASSUME  W.CTL,0
                .BLKB    1          ; NUMBER OF ACTIVE MAPPING POINTERS
        W.USE:  .BLKB    1          ; STATUS OF BLOCK
        W.VBN:  .BLKB    1          ; HIGH BYTE OF 1ST VBN MAPPED BY WINDOW
        W.WISZ: .BLKB    1          ; SIZE IN RTRV PTRS OF WINDOW (7 BITS)
                .BLKW    1          ; LOW ORDER WORD OF 1ST VBN MAPPED
        W.RTRV:                     ; OFFSET TO 1ST RETRIEVAL POINTER IN WINDOW

                .ENDC ;P$$WND       ; END SECONDARY POOL WINDOW CONDITIONAL

                .ENDC ;SYSDEF       ; END SYSDEF CONDITIONAL

                .ENDC ;R$$11M       ; END RSX-11M CONDITIONAL


        ;
        ; LOCKED BLOCK LIST NODE
        ;
                .ASECT
        .=0
000000  L.LNK:  .BLKW    1          ; LINK TO NEXT NODE IN LIST
000002  L.WI1:  .BLKW    1          ; POINTER TO WINDOW FOR FIRST ENTRY
```

```
                .IF DF R$$11D

        L.STD:  .BLKW   1       ; POINTER TO STD OF TASK NODE CHARGED TO
        L.VB1:  .BLKW   2       ; STARTING VBN OF FIRST ENTRY
        L.VB2:  .BLKW   2       ; STARTING VBN OF SECOND ENTRY
        L.CNT:  .BLKB   1       ; COUNT FOR FIRST ENTRY
                .BLKB   1       ; COUNT FOR SECOND ENTRY

                .IFF

000004  L.VB1:  .BLKB   1       ; HIGH ORDER VBN BYTE
000005  L.CNT:  .BLKB   1       ; COUNT FOR ENTRY
000006          .BLKW   1       ; LOW ORDER VBN

                .ENDC ;R$$11D

000010  L.LKSZ:

                .PSECT
```

HDRDF$

```
                HDRDF$

        ;
        ; TASK HEADER OFFSET DEFINITIONS
        ;
                .ASECT
        .=0
000000  H.CSP:  .BLKW   1       ;CURRENT STACK POINTER
000002  H.HDLN: .BLKW   1       ;HEADER LENGTH IN BYTES
000004  H.EFLM: .BLKW   2       ;EVENT FLAG MASK WORD AND ADDRESS
000010  H.CUIC: .BLKW   1       ;CURRENT TASK UIC
000012  H.DUIC: .BLKW   1       ;DEFAULT TASK UIC
000014  H.IPS:  .BLKW   1       ;INITIAL PROCESSOR STATUS WORD (PS)
000016  H.IPC:  .BLKW   1       ;INITIAL PROGRAM COUNTER (PC)
000020  H.ISP:  .BLKW   1       ;INITIAL STACK POINTER (SP)
000022  H.ODVA: .BLKW   1       ;ODT SST VECTOR ADDRESS
000024  H.ODVL: .BLKW   1       ;ODT SST VECTOR LENGTH
000026  H.TKVA: .BLKW   1       ;TASK SST VECTOR ADDRESS
000030  H.TKVL: .BLKW   1       ;TASK SST VECTOR LENGTH
000032  H.PFVA: .BLKW   1       ;POWER FAIL AST CONTROL BLOCK ADDRESS
000034  H.FPVA: .BLKW   1       ;FLOATING POINT AST CONTROL BLOCK ADDRESS
000036  H.RCVA: .BLKW   1       ;RECIEVE AST CONTROL BLOCK ADDRESS
000040  H.EFSV: .BLKW   1       ;EVENT FLAG ADDRESS SAVE ADDRESS
000042  H.FPSA: .BLKW   1       ;POINTER TO FLOATING POINT/EAE SAVE AREA
000044  H.WND:  .BLKW   1       ;POINTER TO NUMBER OF WINDOW BLOCKS
000046  H.DSW:  .BLKW   1       ;TASK DIRECTIVE STATUS WORD
000050  H.FCS:  .BLKW   1       ;FCS IMPURE POINTER
000052  H.FORT: .BLKW   1       ;FORTRAN IMPURE POINTER
000054  H.OVLY: .BLKW   1       ;OVERLAY IMPURE POINTER
000056  H.VEXT: .BLKW   1       ;WORK AREA EXTENSION VECTOR POINTER
000060  H.SPRI: .BLKB   1       ;PRIORITY DIFFERENCE FOR SWAPPING
000061  H.NML:  .BLKB   1       ;NETWORK MAILBOX LUN
000062  H.RRVA: .BLKW   1       ;RECEIVE BY REFERENCE AST CONTROL BLOCK ADDRESS
000064  H.X25:  .BLKB   1       ;FOR USE BY X.25 SOFTWARE
000065          .BLKB   1       ;FIVE RESERVED BYTES
000066          .BLKW   2       ;
000072  H.GARD: .BLKW   1       ;POINTER TO HEADER GUARD WORD
000074  H.NLUN: .BLKW   1       ;NUMBER OF LUN'S
000076  H.LUN:  .BLKW   2       ;START OF LOGICAL UNIT TABLE
```

```
        ;
        ; LENGTH OF FLOATING POINT SAVE AREA
        ;
        H.FPSL=25.*2


        ;
        ; WINDOW BLOCK OFFSETS
        ;
        .=0
000000  W.BPCB: .BLKW    1        ;PARTITION CONTROL BLOCK ADDRESS
000002  W.BLVR: .BLKW    1        ;LOW VIRTUAL ADDRESS LIMIT
000004  W.BHVR: .BLKW    1        ;HIGH VIRTUAL ADDRESS LIMIT
000006  W.BATT: .BLKW    1        ;ADDRESS OF ATTACHMENT DESCRIPTOR
000010  W.BSIZ: .BLKW    1        ;SIZE OF WINDOW IN 32W BLOCKS
000012  W.BOFF: .BLKW    1        ;PHYSICAL MEMORY OFFSET IN 32W BLOCKS
000014  W.BFPD: .BLKB    1        ;FIRST PDR ADDRESS
000015  W.BNPD: .BLKB    1        ;NUMBER OF PDR'S TO MAP
000016  W.BLPD: .BLKW    1        ;CONTENTS OF LAST PDR

000020  W.BLGH:                   ;LENGTH OF WINDOW DESCRIPTOR

                .PSECT
```

HWDDF$

```
        HWDDF$

;
; HARDWARE REGISTER ADDRESSES AND STATUS CODES
;
MPCSR=177746            ;ADDRESS OF PDP-11/70 MEMORY PARITY REGISTER
MPAR=172100             ;ADDRESS OF FIRST MEMORY PARITY REGISTER
PIRQ=177772             ;PROGRAMMED INTERRUPT REQUEST REGISTER
PR0=0                   ;PROCESSOR PRIORITY 0
PR1=40                  ;PROCESSOR PRIORITY 1
PR4=200                 ;PROCESSOR PRIORITY 4
PR5=240                 ;PROCESSOR PRIORITY 5
PR6=300                 ;PROCESSOR PRIORITY 6
PR7=340                 ;PROCESSOR PRIORITY 7
PS=177776               ;PROCESSOR STATUS WORD
SWR=177570              ;CONSOLE SWITCH AND DISPLAY REGISTER
TPS=177564              ;CONSOLE TERMINAL PRINTER STATUS REGISTER


;
; EXTENDED ARITHMETIC ELEMENT REGISTERS
;
        .IF DF E$$EAE

AC=177302               ;ACCUMULATOR
MQ=177304               ;MULTIPLIER-QUOTIENT
SC=177310               ;SHIFT COUNT

        .ENDC ;E$$EAE


;
; MEMORY MANAGEMENT HARDWARE REGISTERS AND STATUS CODES
;
        .IF DF M$$MGE

KDSAR0=172360           ;KERNEL D PAR 0
KDSDR0=172320           ;KERNEL D PDR 0
KISAR0=172340           ;KERNEL I PAR 0
KINAR0=KISAR0           ;KERNEL I PAR 0
KISAR5=172352           ;KERNEL I PAR 5
KINAR5=KISAR5           ;KERNEL I PAR 5
KISAR6=172354           ;KERNEL I PAR 6
KINAR6=KISAR6           ;KERNEL I PAR 6
KISAR7=172356           ;KERNEL I PAR 7
KINAR7=KISAR7           ;KERNEL I PAR 7
KISDR0=172300           ;KERNEL I PDR 0
KISDR6=172314           ;KERNEL I PDR 6
KISDR7=172316           ;KERNEL I PAR 7
SISDR0=172200           ;SUPERVISOR I PDR 0
UDSAR0=177660           ;USER D PAR 0
UDSDR0=177620           ;USER D PDR 0
UISAR0=177640           ;USER I PAR 0
UISAR4=177650           ;USER I PAR 4
UISAR5=177652           ;USER I PAR 5
UISAR6=177654           ;USER I PAR 6
UISAR7=177656           ;USER I PAR 7
UISDR0=177600           ;USER I PDR 0
```

```
UISDR4=177610          ;USER I PDR 4
UISDR5=177612          ;USER I PDR 5
UISDR6=177614          ;USER I PDR 6
UISDR7=177616          ;USER I PDR 7
UBMPR=170200           ;UNIBUS MAPPING REGISTER 0
CMODE=140000           ;CURRENT MODE FIELD OF PS WORD
PMODE=30000            ;PREVIOUS MODE FIELD OF PS WORD
SR0=177572             ;SEGMENT STATUS REGISTER 0
SR3=172516             ;SEGMENT STATUS REGISTER 3

        .ENDC ;M$$MGE


;
; FEATURE SYMBOL DEFINITIONS
;
FE.EXT=1               ;22-BIT EXTENDED MEMORY SUPPORT
FE.MUP=2               ;MULTI-USER PROTECTION SUPPORT
FE.EXV=4               ;EXECUTIVE IS SUPPORTED TO 20K
FE.DRV=10              ;LOADABLE DRIVER SUPPORT
FE.PLA=20              ;PLAS SUPPORT
FE.CAL=40              ;DYNAMIC CHECKPOINT SPACE ALLOCATION
FE.PKT=100             ;PREALLOCATION OF I/O PACKETS
FE.EXP=200             ;EXTEND TASK DIRECTIVE SUPPORTED
FE.LSI=400             ;PROCESSOR IS AN LSI-11
FE.OFF=1000            ;PARENT OFFSPRING TASKING SUPPORTED
FE.FDT=2000            ;FULL DUPLEX TERMINAL DRIVER
FE.X25=4000            ;X.25 COM EXECUTIVE LOADED (1=YES)
FE.DYM=10000           ;DYNAMIC MEMORY ALLOCATION SUPPORTED
FE.CEX=20000           ;COM EXEC IS LOADED
FE.MXT=40000           ;MCR EXIT AFTER EACH COMMAND MODE
FE.NLG=100000          ;LOGINS DISABLED - MULTI-USER SUPPORT


;
; SECOND FEATURE MASK SYMBOL DEFINITIONS
;
F2.DAS=1               ;KERNEL DATA SPACE       (M-PLUS ONLY)
F2.LIB=2               ;SUPERVISOR MODE LIBRARIES       "
F2.MP=4                ;MULTIPROCESSING SUPPORT         "
F2.EVT=10              ;EVENT TRACE SUPPORT             "
F2.ACN=20              ;CPU ACCOUNTING                  "
F2.SDW=40              ;SHADOW RECORDING                "
F2.POL=100             ;SECONDARY POOLS                 "
F2.WND=200             ;SECONDARY POOL FILE WINDOWS     "
F2.DPR=400             ;DIRECTIVE PARTITION SUPPORT
F2.IRR=1000            ;INSTALL, REQUEST AND REMOVE SUPPORT
F2.GGF=2000            ;GROUP GLOBAL EVENT FLAG SUPPORT
F2.RAS=4000            ;RECEIVE/SEND DATA PACKET SUPPORT
F2.AHR=10000           ;ALT. HEADER REFRESH AREAS SUPPORTED
F2.RBN=20000           ;ROUND ROBIN SCHEDULING SUPPORT
F2.SWP=40000           ;EXECUTIVE LEVEL DISK SWAPPING SUPPORT
F2.STP=100000          ;EVENT FLAG MASK IS IN THE TCB (1=YES)
```

```
;
; THIRD FEATURE MASK SYMBOL DEFINITIONS
;
F3.CRA=1                  ;SPONTANEOUS CRASH (1=YES)
F3.NWK=2                  ;SYSTEM HAS NETWORK SUPPORT
F3.EIS=4                  ;SYSTEM REQUIRES THE EXTENDED INST. SET
F3.STM=10                 ;SYSTEM HAS SET SYSTEM TIME DIRECTIVE
F3.UDS=20                 ;USER DATA SPACE (M-PLUS ONLY)
F3.PRO=40                 ;PROTO TCBS OUT OF POOL    "
F3.XHR=100                ;EXTERNAL HEADER SUPPORT  "
F3.AST=200                ;SYSTEM HAS AST SUPPORT
F3.11S=400                ;SYSTEM IS RSX-11S
F3.CLI=1000               ;SYSTEM HAS MULTIPLE CLI SUPPORT
F3.TCM=2000               ;TERMINAL COMMON (M-PLUS ONLY)
F3.PMN=4000               ;POOL MONITORING SUPPORT
F3.WAT=10000              ;WATCHDOG TIMER SUPPORT
F3.RLK=20000              ;'RMS' RECORD LOCKING SUPPORT


;
; HARDWARE FEATURE MASK SYMBOL DEFINITIONS
;
HF.UBM=1                  ;SYSTEM HAS A UNIBUS MAP (1=YES)
HF.EIS=2                  ;SYSTEM HAS EXTENDED INSTRUCTION SET
HF.CIS=200                ;SYSTEM HAS COMMERCIAL INSTRUCTION SET
HF.FPP=100000             ;SYSTEM SUPPORTS FLOATING POINT (1=NO)
```

ITBDF$

```
                ITBDF$   ,,SYSDEF

        ;
        ; INTERRUPT TRANSFER BLOCK (ITB) OFFSET DEFINITIONS
        ;
                .IF DF A$$TRP

                .MCALL  PKTDF$
                PKTDF$             ; DEFINE AST BLOCK OFFSETS

                .ENDC ;A$$TRP

                .ASECT
        .=0
000000  X.LNK:  .BLKW   1          ; LINK WORD FOR ITB LIST STARTING IN TCB
000002  X.JSR:  JSR     R5,@#0     ; CALL $INTSC
000006  X.PSW:  .BLKB   1          ; LOW BYTE OF PSW FOR ISR
000007          .BLKB   1          ; UNUSED
000010  X.ISR:  .BLKW   1          ; ISR ENTRY POINT (APR5 MAPPING)
000012  X.FORK:                    ; FORK BLOCK
000012          .BLKW   1          ; THREAD WORD
000014          .BLKW   1          ; FORK PC
000016          .BLKW   1          ; SAVED R5
000020          .BLKW   1          ; SAVED R4

                .IF DF M$$MGE

        X.REL:  .BLKW   1          ; RELOCATION BASE FOR APR5

                .ENDC ;M$$MGE

        X.DSI:  .BLKW   1          ; ADDRESS OF DIS.INT. ROUTINE
        X.TCB:  .BLKW   1          ; TCB ADDRESS OF OWNING TASK

                .IF NB SYSDEF

                .IF DF A$$TRP

                .BLKW   1          ; A.DQSR FOR AST BLOCK
        X.AST:  .BLKB   A.PRM      ; AST BLOCK

                .ENDC ;A$$TRP

        X.VEC:  .BLKW   1          ; VECTOR ADDRESS (IF AST SUPPORT,
                                   ; THIS IS FIRST AND ONLY AST PARAMETER)
        X.VPC:  .BLKW   1          ; SAVED VECTOR PC
        X.LEN:                     ; LENGTH IN BYTES OF ITB

                .ENDC ;SYSDEF

                .PSECT
```

LCBDF$

```
                LCBDF$

        ;
        ; LOGICAL ASSIGNMENT CONTROL BLOCK
        ;
        ; THE LOGICAL ASSIGNMENT CONTROL BLOCK (LCB) IS USED TO ASSOCIATE A
        ; LOGICAL NAME WITH A PHYSICAL DEVICE UNIT.  LCB'S ARE LINKED TOGETHER
        ; TO FORM THE LOGICAL ASSIGNMENTS OF A SYSTEM.  ASSIGNMENTS MAY BE ON
        ; A SYSTEM WIDE OR LOCAL (TERMINAL) BASIS.
        ;
                .ASECT
        .=0
000000  L.LNK:  .BLKW   1       ;LINK TO NEXT LCB
000002  L.NAM:  .BLKW   1       ;LOGICAL NAME OF DEVICE
000004  L.UNIT: .BLKB   1       ;LOGICAL UNIT NUMBER
000005  L.TYPE: .BLKB   1       ;TYPE OF ENTRY (0=SYSTEM WIDE)
000006  L.UCB:  .BLKW   1       ;TI UCB ADDRESS
000010  L.ASG:  .BLKW   1       ;ASSIGNMENT UCB ADDRESS

000012  L.LGTH=.-L.LNK          ;LENGTH OF LCB

                .PSECT
```

MTADF$

```
                MTADF$

        ;
        ; ANSI MAGTAPE SPECIFIC DATA STRUCTURES
        ;
        ; VOLUME SET CONTROL BLOCK OFFSET DEFININTIONS (VSCB)
        ;
        ; VOLUME SET AND PROCESS CONTROL SECTION
        ;
                .ASECT
        .=0
000000  V.TCNT: .BLKW   1       ;TRANSACTION COUNT
000002  V.TYPE: .BLKB   1       ;VOLUME TYPE DESCRIPTOR
000003  V.VCHA: .BLKB   1       ;VOLUME CHARACTERISTICS
000004  V.LABL: .BLKB   12.     ;FILE SET ID (FIRST SIX BYTES)
000020  V.NXT:  .BLKW   1       ;PTR TO NEXT VSCB NODE
000022  V.MVL:  .BLKW   1       ;PTR TO MOUNTED VOL LIST
000024  V.UVL:  .BLKW   1       ;PTR TO UNMOUNTED VOL LIST
000026  V.ATL:  .BLKW   1       ;ATL ADDR OF ACCESSING TASK TCB IN RSX11M
000030  V.UCB:  .BLKW   1       ;ADDR OF CURRENT UCB OR PUD
000032  V.RVOL: .BLKB   1       ;CURRENT RELATIVE VOL #
000033  V.MOU:  .BLKB   1       ;MOUNT MODE BYTE
000034  V.TCHR: .BLKW   1       ;UINT CHAR. FOR ALL UNITS USED FOR VOL SET
000036  V.SEQN: .BLKW   1       ;CURRENT FILE SEQUENCE #
000040  V.SECN: .BLKW   1       ;CURRENT FILE SECTION #
000042  V.TPOS: .BLKB   1       ;POSITION OF TAPE IN TM'S TO NXT HDR1
000043  V.PSTA: .BLKB   1       ;PROCESS STATUS BYTE
000044  V.TIMO: .BLKW   1       ;BLOCKED PROCESS TIMEOUT COUNTER
000046  V.STAT: .BLKW   3       ;STATUS WORDS USED BY COMMAND EXECUTION MODULES
000054  V.TRTB: .BLKB   1       ;TRANSLATION CONTROL BYTE
000055  V.EFTV: .BLKB   1       ;FOR MAG TO RETURN IE.EOF, EOT, EOV


        ;
        ; LABEL DATA SECTION
        ;
000056  V.BLKL: .BLKW   1       ;BLOCK LENGTH
000060  V.RECL: .BLKW   1       ;RECORD LENGTH
000062  V.FNAM: .BLKW   3       ;FILE NAME
000070  V.FTYP: .BLKW   1       ;FILE TYPE
000072  V.FVER: .BLKW   1       ;FILE VERSION #
000074  V.CDAT: .BLKW   2       ;CREATION DATE
000100  V.EDAT: .BLKW   2       ;EXPRIATION DATE
000104  V.BLKC: .BLKW   2       ;BLOCK COUNT FOR FILE SECTION
000110  V.RTYP: .BLKB   1       ;RECORD TYPE
000111  V.FATT: .BLKB   1       ;FILE ATTRIBUTES FOR CARRIAGE CONTROL
000112          .BLKB   30.     ;REMAINDER OF FILE ATTRIBUTES
```

```
        ;
        ; NULL WINDOW SECTION
        ;
000150  V.WIND: .BLKW   4.      ;NULL WINDOW
000160  V.MST2: .BLKW   1       ;MAGTAPE STATUS BITS
000162  V.FABY: .BLKB   1       ;FILE ACCESSIBILITY BYTE (HDR1)
000163          .BLKB   1       ;SPARE
000164  V.ANSN: .BLKB   17.     ;ANSI 17 CHARACTER FILE NAME
000205  V.BOFF: .BLKB   1.      ;BUFFER OFFSET
000206  V.DENS: .BLKB   1.      ;REQUESTED UNIT DENSITY
000207  V.DRAT: .BLKB   1.      ;DEFAULT RECORD ATTRIBUTES
000210  V.DBLK: .BLKW   1.      ;DEFAULT BLOCK SIZE
000212  V.DREC: .BLKW   1.      ;DEFAULT RECORD SIZE

000214  S.VSCB=.                ;SIZE OF VSCB

                .PSECT

        ;
        ; DEFINE OFFSETS INTO NULL WINDOW SECTION
        ;
                .ASECT
        .=0
000000  W.CTL:  .BLKW   1       ;CONTROL WORD IN WINDOW
        V.WINC=V.WIND+W.CTL     ;CNTRL WORD IN NULL WINDOW
                                ;RELATIVE TO THE VSCB

                .PSECT

        ;
        ; MOUNTED VOLUME LIST OFFSET DEFININTIONS (MVL)
        ;
                .ASECT
        .=0

                .IF DF R$$11M

000000  M.NXT:  .BLKW   1       ;PTR TO NXT MVL NODE (11M)

                .ENDC ;R$$11M

000002  M.UIC:  .BLKW   1       ;OWNER UIC FROM RVOL #1
000004  M.CH:   .BLKW   1       ;U.CH/U.VP (11D)
000006  M.PROT: .BLKW   1       ;PROTECTION U.AR IN 11D

                .IF NDF R$$11M

                .BLKW   2       ;ACP WORDS 11D
        M.NXT:  .BLKW   1       ;PTR TO NEXT MVL NODE (11D)

                .ENDC ;R$$11M

000010  M.RVOL: .BLKB   1       ;RELATIVE VOL # OF MOUNTED VOLUME
000011  M.STAT: .BLKB   1       ;VOLUME STATUS
000012  M.VIDP: .BLKW   1       ;VOLUME ID POINTER
000014  M.UCB:  .BLKW   1       ;ADDR OF ASSOC UCB OR PUD

000016  S.MVL=.                 ;SIZE OF MVL NODE

                .PSECT
```

```
        ;
        ; UNMOUNTED VOLUME AND VOLUME  LIST OFFSET DEFINITIONS (UVL)
        ;
                .ASECT
        .=0
000000  L.NXT:  .BLKW   1       ;PTR TO NXT UVL NODE
000002  L.VOL1: .BLKB   1       ;REL VOL # OF 1'ST VOL IN NODE
000003  L.VOL2: .BLKB   1       ;REL VOL # OF 2'ND VOL IN NODE
000004  L.VID1: .BLKB   6       ;VOL ID OF 1'ST VOL IN NODE
000012  L.VID2: .BLKB   6       ;VOL ID OF 2'ND VOL IN NODE

000020  S.UVL=.                 ;SIZE OF UVL NODE

                .PSECT

        ;
        ; SYSTEM DATA STRUCTURE CONTENT VALUES
        ;

        ;
        ; VSCB VALUES
        ;
        ; V.MOU VALUES
        ;
        VM.OLD  =       200     ;OLD .FL300 VOLUME -- VM.BYP WILL ALSO BE SET
        VM.BYP  =       100     ;BYPASS LABEL PROCESSING
        VM.ULB  =       40      ;UNLABELED TAPE
        VM.FSC  =       20      ;OVERRIDE FILE SET ID CHECK
        VM.EXC  =       10      ;OVERRIDE EXPRIATION DATE CHECK


        ;
        ; V.MST2 VALUES
        ;
        V2.INI  =       1       ;MAG WANTS US TO INITIALIZE NEXT OUTPUT
        V2.XH2  =       2       ;THIS FILE HAS NO HDR2, DON'T WRITE EOF2
        V2.XH3  =       4       ;THIS FILE HAS NO HDR3, DON'T WRITE EOF3
        V2.NH3  =       10      ;DON'T WRITE HDR3/EOX3 LABELS
        V2.OAC  =       20      ;OVERRIDE FILE/VOLUME ACCESSIBILITY


        ;
        ; V.PSTA VALUES - UNBLOCKED TRANSITION STATE
        ;
        VP.RM   =       2       ;READ DATA MODE
        VP.WM   =       4       ;WRITE DATA MODE
        VP.UCM  =       6       ;UNLABELLED CREATE POSITIONING MODE
        VP.SM   =       10      ;SEARCH MODE
        VP.MOU  =       20      ;MOUNT MODE
        VP.RWD  =       40      ;REWIND OR VOL VERIFICATION WAIT
        VP.VFY  =       VP.RWD
        VP.POS  =       100     ;PROCESS IN POSITIONING MODE
                                ;(MULTI-SECTION FILE)


        ;
        ; BLOCKED STATE = -(UNBLOCKED TRANSITION STATE VALUES)
        ;
        ; PROCESS TIMED OUT BIT 0 = 1
        ;
        VP.TO=1
```

```
;
; NULL WINDOW CONTROL BIT DEFINITIONS
;
WI.RDV  =       400     ;ACCESSED FOR READ
WI.WRV  =       1000    ;ACCESSED FOR WRITE
WI.EXT  =       2000    ;ACCESSED FOR EXTEND
WI.LCK  =       4000    ;LOCKED


;
; MVL VALUES IN THE M.STAT FIELD
;
MS.VER  =       200     ;VOL ID NOT VERIFIED
MS.RID  =       1       ;VOL ID TO BE READ NOT CHECKED
MS.NMO  =       2       ;MOUNT MESSAGE NOT GIVEN YET
MS.TMO  =       4       ;ONE TIMEOUT ALREADY EXPRIED
MS.EXP  =       10      ;EXPIRATION DATE MESSAGE GIVEN


;
; MISC BITS USED IN MOUNT (STORED IN V.STS)
;
MO.OVR  =       1       ;OVER RIDE VOL NAME SWITCH
MO.UIC  =       2       ;EXPLICIT UIC GIVEN
MO.PRO  =       4       ;EXPLICIT PROTECTION GIVEN
MO.160  =       10      ;1600 BPI SPECIFIED
```

PCBDF$

```
                PCBDF$   ,,SYSDEF

        ;
        ; PARTITION CONTROL BLOCK OFFSET DEFINITIONS
        ;
                .ASECT
        .=0
000000  P.LNK:  .BLKW   1          ;LINK TO NEXT PARTITION PCB
000002  P.PRI:  .BLKB   1          ;PRIORITY OF PARTITION
000003  P.IOC:  .BLKB   1          ;I/O + I/O STATUS BLOCK COUNT
000004  P.NAM:  .BLKW   2          ;PARTITION NAME IN RAD50
000010  P.SUB:  .BLKW   1          ;POINTER TO NEXT SUBPARTITION
000012  P.MAIN: .BLKW   1          ;POINTER TO MAIN PARTITION

                .IF NB   SYSDEF

                .IF NDF M$$MGE

        P.HDR:                     ;POINTER TO HEADER CONTROL BLOCK

                .ENDC   ;M$$MGE

                .IFTF

000014  P.REL:  .BLKW   1          ;STARTING PHYSICAL ADDRESS OF PARTITION
000016  P.BLKS:
000016  P.SIZE: .BLKW   1          ;SIZE OF PARTITION IN:
                                   ;   UNMAPPED SYSTEMS - BYTES
                                   ;   MAPPED SYSTEMS   - 32 WORD BLOCKS
000020  P.WAIT: .BLKW   1          ;PARTITION WAIT QUEUE LISTHEAD (2 WORDS)
000022  P.SWSZ: .BLKW   1          ;PARTITION SWAP SIZE (SYSTEM ONLY)
000024  P.BUSY: .BLKB   2          ;PARTITION BUSY FLAGS
000026  P.OWN:
000026  P.TCB:  .BLKW   1          ;TCB ADDRESS OF OWNER TASK
000030  P.STAT: .BLKW   1          ;PARTITION STATUS FLAGS

                .IFT

                .IF DF   M$$MGE

        P.HDR:  .BLKW   1          ;POINTER TO HEADER CONTROL BLOCK

                .ENDC   ;M$$MGE

        P.PRO:  .BLKW   1          ;PROTECTION WORD [DEWR,DEWR,DEWR,DEWR]
        P.ATT:  .BLKW   2          ;ATTACHMENT DESCRIPTOR LISTHEAD

                .IF NDF P$$LAS

        P.LGTH=P.PRO               ;LENGTH OF PARTITION CONTROL BLOCK
```

```
        .IFF

P.LGTH=.                        ;LENGTH OF PARTITION CONTROL BLOCK

        .ENDC   ;P$$LAS

        .IFF

        .PSECT

;
; PARTITION STATUS WORD BIT DEFINITIONS
;
PS.OUT=100000                   ;PARTITION IS OUT OF MEMORY(1=YES)
PS.CKP=40000                    ;PARTITION CHECKPOINT IN PROGRESS (1=YES)
PS.CKR=20000                    ;PARTITION CHECKPOINT IS REQUESTED (1=YES)
PS.CHK=10000                    ;PARTITION IS NOT CHECKPOINTABLE (1=YES)
PS.FXD=4000                     ;PARTITION IS FIXED (1=YES)
PS.PER=2000                     ;PARITY ERROR IN PARTITION (1=YES)
PS.LIO=1000                     ;MARKED BY SHUFFLER FOR LONG I/O (1=YES)
PS.NSF=400                      ;PARTITION IS NOT SHUFFLEABLE (1=YES)
PS.COM=200                      ;LIBRARY OR COMMON BLOCK (1=YES)
PS.PIC=100                      ;POSITION INDEPENDENT LIBRARY OR COMMON (1=YES)
PS.SYS=40                       ;SYSTEM CONTROLLED PARTITION (1=YES)
PS.DRV=20                       ;DRIVER IS LOADED IN PARTITION (1=YES)
PS.DEL=10                       ;PARTITION SHOULD BE DELETED WHEN NOT ATTACHED
                                ;(1=YES)
PS.APR=7                        ;STARTING APR NUMBER MASK


;
; ATTACHMENT DESCRIPTOR OFFSETS
;
        .ASECT
.=0
000000  A.PCBL: .BLKW   1       ;PCB ATTACHMENT QUEUE THREAD WORD
000002  A.PRI:  .BLKB   1       ;PRIORITY OF ATTACHED TASK
000003  A.IOC:  .BLKB   1       ;I/O COUNT THROUGH THIS DESCRIPTOR
000004  A.TCB:  .BLKW   1       ;TCB ADDRESS OF ATTACHED TASK
000006  A.TCBL: .BLKW   1       ;TCB ATTACHMENT QUEUE THREAD WORD
000010  A.STAT: .BLKB   1       ;STATUS BYTE
000011  A.MPCT: .BLKB   1       ;MAPPING COUNT OF TASK THRU THIS DESCRIPTOR
000012  A.PCB:  .BLKW   1       ;PCB ADDRESS OF ATTACHED TASK

000014  A.LGTH=.                ;LENGTH OF ATTACHMENT DESCRIPTOR


;
; ATTACHMENT DESCRIPTOR STATUS BYTE BIT DEFINITIONS
;
        .PSECT
AS.DEL=10                       ;TASK HAS DELETE ACCESS (1=YES)
AS.EXT=4                        ;TASK HAS EXTEND ACCESS (1=YES)
AS.WRT=2                        ;TASK HAS WRITE ACCESS (1=YES)
AS.RED=1                        ;TASK HAS READ ACCESS (1=YES)

        .ENDC   ;SYSDEF
```

PKTDF$

```
                PKTDF$

        ;
        ; ASYNCHRONOUS SYSTEM TRAP CONTROL BLOCK OFFSET DEFINITIONS
        ;
        ; SOME POSITIONAL DEPENDENCIES BETWEEN THE OCB AND THE AST CONTROL
        ; BLOCK ARE RELIED UPON IN THE ROUTINE $FINXT IN THE MODULE SYSXT.
        ;
                .ASECT
        .=177774
177774  A.KSR5: .BLKW   1           ;SUBROUTINE KISAR5 BIAS (A.CBL=0)
177776  A.DQSR: .BLKW   1           ;DEQUEUE SUBROUTINE ADDRESS (A.CBL=0)
000000          .BLKW   1           ;AST QUEUE THREAD WORD
000002  A.CBL:  .BLKW   1           ;LENGTH OF CONTROL BLOCK IN BYTES
                                    ;IF A.CBL = 0, THE AST CONTROL BLOCK IS
                                    ;TO BE DEALLOCATED BY THE DEQUEUE SUBROUTINE
                                    ;POINTED TO BY A.DQSR MAPPED VIA APR 5
                                    ;VALUE A.KSR5. THIS IS CURRENTLY USED ONLY
                                    ;BY THE FULL DUPLEX TERMINAL DRIVER FOR
                                    ;UNSOLICITED CHARACTER ASTS.
                                    ;IF THE LOW BYTE OF A.CBL = 0, AND THE
                                    ;HIGH BYTE IS NOT = 0, THE AST CONTROL BLOCK
                                    ;IS A SPECIFIABLE AST, WITH LENGTH, C.LGTH.
                                    ;IF THE HIGH BYTE OF A.CBL = 0 AND THE LOW
                                    ;BYTE > 0, THEN THE LOW BYTE IS THE LENGTH
                                    ;OF THE AST CONTROL BLOCK.  IF THE HIGH BYTE
                                    ;OF A.CBL = 0 AND THE LOW BYTE IS NEGATIVE,
                                    ;THIS IS A KERNEL AST. SEE BELOW FOR
                                    ;A DESCRIPTION OF A.CBL FOR KERNEL ASTS.
000004  A.BYT:  .BLKW   1           ;NUMBER OF BYTES TO ALLOCATE ON TASK STACK
000006  A.AST:  .BLKW   1           ;AST TRAP ADDRESS
000010  A.NPR:  .BLKW   1           ;NUMBER OF AST PARAMETERS
000012  A.PRM:  .BLKW   1           ;FIRST AST PARAMETER


        ;
        ; THE SPECIFIABLE AST CODES MUST NOT BE 0.
        ;
        AS.FPA=1                    ;CODE FOR FLOATING POINT AST
        AS.RCA=2                    ;CODE FOR RECEIVE DATA AST
        AS.RRA=3                    ;CODE FOR RECEIVE BY REFERENCE AST
        AS.PFA=4                    ;CODE FOR POWERFAIL AST
        AS.REA=5                    ;CODE FOR REQUESTED EXIT (ABORT) AST
        AS.CAA=6                    ;CODE FOR COMMAND ARRIVAL AST FOR CLIS


        ;
        ; ABORTER SUBCODES FOR ABORT AST (AS.REA) TO BE RETURNED ON USER'S
        ; STACK
        ;
        AB.NPV=1                    ;ABORTER IS NONPRIVILEGED (1=YES)
        AB.TYP=2                    ;ABORT FROM DIRECTIVE (0=YES)
                                    ;ABORT FROM CLI COMMAND (1=YES)
```

```
        ;
        ; KERNEL AST CONTROL BLOCK DEFINITIONS
        ;
        ; THE LOW BYTE OF A.CBL FOR A KERNEL AST HAS THE FOLLOWING FORMAT:
        ;
        ;       BIT #200 ALWAYS EQUALS 1
        ;       BIT #100 IS ZERO IF $SGFIN MUST BE CALLED DURING AST PROCESSING
        ;       THE REMAINING SIX BITS ARE USED AS THE KERNEL AST TYPE FIELD
        ;
        ; BECAUSE THERE ARE ONLY 6 BITS AVAILABLE TO THE KERNEL AST
        ; INDEX FIELD, ONLY (2**6)-1 KERNEL AST TYPES ARE POSSIBLE.
        ;
        AK.BUF=200              ;BUFFERED I/O COMPLETION AST
        AK.OCB=201              ;OFFSPRING EXIT
        AK.GBI=202              ;GENERAL BUFFERED I/O AST
        AK.TBT=203              ;TASK FORCED T-BIT TRAP AST
        AK.DIO=204              ;DELAYED I/O (M-PLUS COMPATIBLE)


        ;
        ; OFFSPRING CONTROL BLOCK DEFINITIONS
        ;
        ; SOME POSITIONAL DEPENDENCIES EXIST BETWEEN THE OCB AND THE AST
        ; CONTROL BLOCK IN ROUTINE $FINXT IN MODULE SYSXT
        ;
        .=0
000000  O.LNK:  .BLKW   1       ;OCB LINK WORD
000002  O.MCRL: .BLKW   1       ;ADDRESS OF MCR COMMAND LINE
000004  O.PTCB: .BLKW   1       ;PARENT TCB ADDRESS
000006  O.AST:  .BLKW   1       ;EXIT AST ADDRESS
000010  O.EFN:  .BLKW   1       ;EXIT EVENT FLAG
000012  O.ESB:  .BLKW   1       ;EXIT STATUS BLOCK VIRTUAL ADDRESS
000014  O.STAT: .BLKW   8.      ;EXIT STATUS BUFFER

000034  O.LGTH=.                ;LENGTH OF OCB


        ;
        ; I/O PACKET OFFSET DEFINITIONS
        ;
                .ASECT
        .=0
000000  I.LNK:  .BLKW   1       ;I/O QUEUE THREAD WORD
000002  I.PRI:  .BLKB   1       ;REQUEST PRIORITY
000003  I.EFN:  .BLKB   1       ;EVENT FLAG NUMBER
000004  I.TCB:  .BLKW   1       ;TCB ADDRESS OF REQUESTOR
000006  I.LN2:  .BLKW   1       ;POINTER TO SECOND LUN WORD
000010  I.UCB:  .BLKW   1       ;POINTER TO UNIT CONTROL BLOCK
000012  I.FCN:  .BLKW   1       ;I/O FUNCTION CODE
000014  I.IOSB: .BLKW   1       ;VIRTUAL ADDRESS OF I/O STATUS BLOCK
000016          .BLKW   1       ;I/O STATUS BLOCK RELOCATON BIAS
000020          .BLKW   1       ;I/O STATUS BLOCK ADDRESS
000022  I.AST:  .BLKW   1       ;AST SERVICE ROUTINE ADDRESS
000024  I.PRM:  .BLKW   1       ;RESERVED FOR MAPPING PARAMETER #1
000026          .BLKW   6       ;PARAMETERS 1 TO 6
000042          .BLKW   1       ;USER MODE DIAGNOSTIC PARAMETER WORD

000044  I.ATTL=.                ;MINIMUM LENGTH OF I/O PACKET (USED BY
                                ;FILE SYSTEM TO CALCULATE MAXIMUM
                                ;NUMBER OF ATTRIBUTES)
000044  I.LGTH=.                ;LENGTH OF I/O REQUEST CONTROL BLOCK
```

```
        ;
        ; GROUP GLOBAL EVENT FLAG CONTROL BLOCK OFFSETS
        ;
        .=0
000000  G.LNK:  .BLKW   1       ;LINK WORD
000002  G.GRP:  .BLKB   1       ;GROUP NUMBER
000003  G.STAT: .BLKB   1       ;STATUS BYTE
000004  G.CNT:  .BLKW   1       ;ACCESS COUNT
000006  G.EFLG: .BLKW   2       ;EVENT FLAGS

000012  G.LGTH=.                ;LENGTH OF GROUP GLOBAL CONTROL BLOCK


        ;
        ; STATUS BYTE DEFINITIONS
        ;
        GS.DEL=1                ;GROUP MARKED FOR DELETE


        ;
        ; EXECUTIVE POOL MONITOR CONTROL FLAGS
        ;

        ;
        ; $POLST IS THE SYNCHRONIZATION WORD BETWEEN THE EXEC AND POOL MONITOR
        ;
        PC.HIH=1                ;HIGH POOL LIMIT CROSSED (1=YES)
        PC.LOW=2                ;LOW POOL LIMIT CROSSED (1=YES)
        PC.ALF=4                ;POOL ALLOCATION FAILURE (1=YES)
        PC.NRM=PC.HIH*400       ;POOL TASK INHIBIT BIT FOR HIGH POOL
        PC.ALM=PC.LOW*400       ;POOL TASK INHIBIT BIT FOR LOW POOL

        ;
        ; $POLFL IS THE POOL USAGE CONTROL WORD
        ;
        PF.INS=40               ;REJECT NONPRIVILEGED INS/RUN/REM
        PF.LOG=100              ;LOGINS ARE DISABLED
        PF.REQ=200              ;STALL REQUEST OF NONPRIV. TASKS
        PF.ALL=177777           ;TAKE ALL POSSIBLE ACTIONS TO SAVE POOL


        ;
        ; CLI PARSER BLOCK (CPB) DEFINITIONS
        ;
        .=0
000000  C.PTCB: .BLKW   1       ;ADDRESS OF CLI'S TCB
000002  C.PNAM: .BLKW   2       ;CLI NAME
000006  C.PSTS: .BLKW   1       ;STATUS MASK
000010  C.PDPL: .BLKB   1       ;LENGTH OF DEFAULT PROMPT
000011  C.PCPL: .BLKB   1       ;LENGTH OF CNTRL/C PROMPT
000012  C.PRMT:                 ;START OF ASCII PROMPT STRINGS
                                ;THE DEFAULT STRING IS CONCANTENATED
                                ;WITH THE ^C STRING
```

```
        ;
        ; STATUS BIT DEFINITIONS
        ;
        CP.NUL=1                ;PASS EMPTY COMMAND LINES TO CLI
        CP.MSG=2                ;CLI DESIRES SYSTEM MESSAGES
        CP.LGO=4                ;CLI WANTS COMMANDS FROM LOGGED OFF TTYS
        CP.DSB=10               ;CLI IS DISABLED
        CP.PRV=20               ;USER MUST BE PRIV TO SET TTY TO THIS CLI
        CP.SGL=40               ;DON'T HANDLE CONTINUATIONS (M-PLUS ONLY)
        CP.NIO=100              ;MCR..., HEL, BYE DO NO I/O TO TTY
                                ;HEL, BYE ALSO DO NOT SET CLI ETC.
        CP.RST=200              ;ABILITY TO SET TO THIS CLI IS RESTRICTED
                                ;TO THE CLI ITSELF
        CP.EXT=400              ;PASS TASK EXIT PROMPT REQUESTS TO CLI


        ;
        ; IDENTIFIER CODES FOR SYSTEM TO CLI MESSAGES.
        ;
        ; CODES 0 - 127. ARE RESERVED FOR USE BY DIGITAL,
        ; CODES 128. - 255. ARE RESERVED FOR USE BY CUSTOMERS
        ;
        CM.INE=1                ;CLI INITIALIZED ENABLED
        CM.IND=2                ;CLI INITIALIZED DISABLED
        CM.CEN=3                ;CLI ENABLED
        CM.CDS=4                ;CLI DISABLED
        CM.ELM=5                ;CLI BEING ELIMINATED
        CM.EXT=6                ;CLI MUST EXIT IMMEDIATELY
        CM.LKT=7                ;NEW TERMINAL LINKED TO CLI
        CM.RMT=8.               ;TERMINAL REMOVED FROM CLI
        CM.MSG=9.               ;GENERAL MESSAGE TO CLI


        ;
        ; ANCILLARY CONTROL BLOCK (ACB) DEFINITIONS
        ;
        .=0
000000  A.REL:  .BLKW   1       ;ACD RELOCATION BIAS
000002  A.DIS:  .BLKW   1       ;ACD DISPATCH TABLE POINTER
000004  A.MAS:  .BLKW   1       ;ACD FUNCTION MASK
000006  A.NUM:  .BLKB   1       ;ACD IDENTIFICATION NUMBER
000007          .BLKB   1       ;RESERVED
000010  A.LIN:  .BLKW   1       ;ACD LINK WORD
000012  A.ACC:  .BLKB   1       ;ACD ACCESS COUNT
000013  A.STA:  .BLKB   1       ;ACD STATUS BYTE

000014  A.LEN1=.                ;LENGTH OF PROTOTYPE ACB


        .=A.LIN                 ;FULL ACB OVERLAPS PROTOTYPE ACB
000010  A.IMAP: .BLKW   1       ;ACD INTERRUPT BUFFER RELOCATION BIAS
000012  A.IBUF: .BLKW   1       ;ACD INTERRUPT BUFFER ADDRESS
000014  A.ILEN: .BLKW   1       ;ACD INTERRUPT BUFFER LENGTH
000016  A.SMAP: .BLKW   1       ;ACD SYSTEM STATE BUFFER RELOCATION BIAS
000020  A.SBUF: .BLKW   1       ;ACD SYSTEM STATE BUFFER ADDRESS
000022  A.SLEN: .BLKW   1       ;ACD SYSTEM STATE BUFFER LENGTH
000024  A.IOS:  .BLKW   2       ;ACD I/O STATUS
000030  A.RES:  .BLKW   1       ;RESERVED FOR USE BY THE ACD

000032  A.LEN2=.                ;LENGTH OF FULL ACB
```

```
        ;
        ; DEFINE THE FLAG VALUES IN THE OFFSET U.AFLG
        ;
        UA.ACC=1                ;ACCEPT THIS CHARACTER
        UA.PRO=2                ;PROCESS THIS CHARACTER
        UA.ECH=4                ;ECHO THIS CHARACTER
        UA.TYP=10               ;FORCE THIS CHARACTER INTO TYPEAHEAD
        UA.SPE=20               ;THIS CHARACTER HAS A SPECIAL ECHO
        UA.PUT=40               ;PUT THIS CHARACTER IN THE INPUT BUFFER
        UA.CAL=100              ;CALL THE ACD BACK AFTER THE TRANSFER
        UA.COM=200              ;COMPLETE THE INPUT REQUEST
        UA.ALL=400              ;ALLOW PROCESSING OF THIS I/O REQUEST
        UA.TRA=1000             ;TRANSFER CHARS. WHEN I/O COMPLETES


        ;
        ; DEFINE THE ACD ENTRY POINTS (OFFSETS INTO THE DISPATCH TABLE)
        ;
        .=0
000000  A.ACCE: .BLKW   1       ;I/O REQUEST ACCEPTANCE ENTRY POINT
000002  A.DEQU: .BLKW   1       ;I/O REQUEST DEQUEUE ENTRY POINT
000004  A.POWE: .BLKW   1       ;POWER FAILURE ENTRY POINT
000006  A.INPU: .BLKW   1       ;INPUT COMPLETION ENTRY POINT
000010  A.OUTP: .BLKW   1       ;OUTPUT COMPLETION ENTRY POINT
000012  A.CONN: .BLKW   1       ;CONNECTION ENTRY POINT
000014  A.DISC: .BLKW   1       ;DISCONNECTION ENTRY POINT
000016  A.RECE: .BLKW   1       ;INPUT CHARACTER RECEPTION ENTRY POINT
000020  A.PROC: .BLKW   1       ;INPUT CHARACTER PROCESSING ENTRY POINT
000022  A.CALL: .BLKW   1       ;CALL ACD BACK AFTER TRANSFER ENTRY POINT


        ;
        ; DEFINE THE STATUS BITS IN A.STA OF THE PROTOTYPE ACB
        ;
        AS.DEL=1                ;ACD IS MARKED FOR DELETE
        AS.DIS=2                ;ACD IS DISABLED

                .PSECT
```

SCBDF$

```
                SCBDF$  ,,SYSDEF

        ;
        ; STATUS CONTROL BLOCK
        ;
        ; THE STATUS CONTROL BLOCK (SCB) DEFINES THE STATUS OF A DEVICE
        ; CONTROLLER.  THERE IS ONE SCB FOR EACH CONTROLLER IN A SYSTEM.
        ; THE SCB IS POINTED TO BY UNIT CONTROL BLOCKS. TO EXPAND ON THE
        ; TELETYPE EXAMPLE ABOVE, EACH TELETYPE INTERFACED VIA A DL11-A
        ; WOULD HAVE A SCB SINCE EACH DL11-A IS AN INDEPENDENT INTERFACE
        ; UNIT.  THE TELETYPES INTERFACED VIA THE DH11 WOULD ALSO EACH HAVE
        ; AN SCB SINCE THE DH11 IS A SINGLE CONTROLLER BUT MULTIPLEXES MANY
        ; UNITS IN PARALLEL.
        ;
                .ASECT
        .=177772
177772  S.RCNT: .BLKB   1       ;NUMBER OF REGISTERS TO COPY ON ERROR
177773  S.ROFF: .BLKB   1       ;OFFSET TO FIRST DEVICE REGISTER
177774  S.BMSV: .BLKW   1       ;SAVED I/O ACTIVE BITMAP AND POINTER TO EMB
177776  S.BMSK: .BLKW   1       ;DEVICE I/O ACTIVE BIT MASK
000000  S.LHD:  .BLKW   2       ;CONTROLLER I/O QUEUE LISTHEAD
000004  S.PRI:  .BLKB   1       ;DEVICE PRIORITY
000005  S.VCT:  .BLKB   1       ;INTERRUPT VECTOR ADDRESS /4
000006  S.CTM:  .BLKB   1       ;CURRENT TIMEOUT COUNT
000007  S.ITM:  .BLKB   1       ;INITIAL TIMEOUT COUNT
000010  S.CON:  .BLKB   1       ;CONTROLLER INDEX
000011  S.STS:  .BLKB   1       ;CONTROLLER STATUS (0=IDLE,1=BUSY)
000012  S.CSR:  .BLKW   1       ;ADDRESS OF CONTROL STATUS REGISTER
000014  S.PKT:  .BLKW   1       ;ADDRESS OF CURRENT I/O PACKET
000016  S.FRK:  .BLKW   1       ;FORK BLOCK LINK WORD
000020  S.DMCS:                 ;DM11-BB CSR FOR FDX TTDRV
000020          .BLKW   1       ;FORK-PC
000022          .BLKW   1       ;FORK-R5
000024          .BLKW   1       ;FORK-R4

                .IF NB  SYSDEF

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

                .BLKW   1       ;FORK-DRIVER RELOCATION BASE

                .ENDC   ;L$$DRV & M$$MGE

        S.CCB:                  ;MIXED MASSBUS CHANNEL CONTROL BLOCK
        S.MPR:  .BLKW   6       ;11/70 EXTENDED MEMORY UNIBUS DEVICE C-BLOCK
                .BLKW   1       ;BUFFER WORD
        S.UMHD: .BLKW   2       ;LIST HEAD FOR UMR ASSIGNMENT BLOCK(S)
        S.UMCT: .BLKW   1       ;COUNT OF AVAILABLE UMR ASSIGNMENT BLOCK(S)

                .IFF

                .PSECT
```

```
        ;
        ; STATUS CONTROL BLOCK PRIORITY BYTE CONDITION CODE STATUS BIT
        ; DEFINITIONS
        ;
        SP.EIP=1                ;ERROR IN PROGRESS (1=YES)
        SP.ENB=2                ;ERROR LOGGING ENABLED (0=YES)
        SP.LOG=4                ;ERROR LOGGING AVAILABLE (1=YES)
        SPARE=10                ;SPARE BIT


        ;
        ; MAPPING ASSIGNMENT BLOCK (FOR UNIBUS MAPPING REGISTER ASSIGNMENT)
        ;
                .ASECT
        .=0
000000  M.LNK:  .BLKW   1       ;LINK WORD
000002  M.UMRA: .BLKW   1       ;ADDRESS OF FIRST ASSIGNED UMR
000004  M.UMRN: .BLKW   1       ;NUMBER OF UMR'S ASSIGNED * 4
000006  M.UMVL: .BLKW   1       ;LOW 16 BITS MAPPED BY 1ST ASSIGNED UMR
000010  M.UMVH: .BLKB   1       ;HIGH 2 BITS MAPPED IN BITS 4 AND 5
000011  M.BFVH: .BLKB   1       ;HIGH 6 BITS OF PHYSICAL BUFFER ADDRESS
000012  M.BFVL: .BLKW   1       ;LOW 16 BITS OF PHYSICAL BUFFER ADDRESS

000014  M.LGTH=.                ;LENGTH OF MAPPING ASSIGNMENT BLOCK

                .ENDC   ;SYSDEF

                .PSECT
```

TCBDF$

```
                TCBDF$  ,,SYSDEF

        ;
        ; TASK CONTROL BLOCK OFFSET AND STATUS DEFINITIONS
        ;
        ; TASK CONTROL BLOCK
        ;
                .ASECT
        .=0
000000  T.LNK:  .BLKW   1       ;UTILITY LINK WORD
000002  T.PRI:  .BLKB   1       ;TASK PRIORITY
000003  T.IOC:  .BLKB   1       ;I/O PENDING COUNT
000004  T.CPCB: .BLKW   1       ;POINTER TO CHECKPOINT PCB
000006  T.NAM:  .BLKW   2       ;TASK NAME IN RAD50
000012  T.RCVL: .BLKW   2       ;RECEIVE QUEUE LISTHEAD
000016  T.ASTL: .BLKW   2       ;AST QUEUE LISTHEAD
000022  T.EFLG: .BLKW   2       ;TASK LOCAL EVENT FLAGS 1-32
000026  T.UCB:  .BLKW   1       ;UCB ADDRESS FOR PSEUDO DEVICE 'TI'
000030  T.TCBL: .BLKW   1       ;TASK LIST THREAD WORD
000032  T.STAT: .BLKW   1       ;FIRST STATUS WORD (BLOCKING BITS)
000034  T.ST2:  .BLKW   1       ;SECOND STATUS WORD (STATE BITS)
000036  T.ST3:  .BLKW   1       ;THIRD STATUS WORD (ATTRIBUTE BITS)
000040  T.DPRI: .BLKB   1       ;TASK'S DEFAULT PRIORITY
000041  T.LBN:  .BLKB   3       ;LBN OF TASK LOAD IMAGE
000044  T.LDV:  .BLKW   1       ;UCB ADDRESS OF LOAD DEVICE
000046  T.PCB:  .BLKW   1       ;PCB ADDRESS OF TASK PARTITION
000050  T.MXSZ: .BLKW   1       ;MAXIMUM SIZE OF TASK IMAGE (MAPPED ONLY)
000052  T.ACTL: .BLKW   1       ;ADDRESS OF NEXT TASK IN ACTIVE LIST
000054  T.SAST: .BLKW   1       ;SPECIFIED AST LISTHEAD
000056          .BLKB   1       ;UNUSED BYTE
000057  T.TIO:  .BLKB   1       ;BUFFERED I/O COUNT
000060  T.TKSZ: .BLKW   1       ;TASK SIZE (FROM L$BLDZ IN LABEL BLK) IN:
                                ;    UNMAPPED SYSTEMS - BYTES
                                ;    MAPPED SYSTEMS   - 32 WORD BLOCKS
                                ;TASK SIZE (FROM L$BMXZ IN LABEL BLK)
                                ;FOR RSX11S SYSTEMS ONLY
                                ;    MAPPED SYSTEMS   - 32 WORD BLOCKS
                                ;    UNMAPPED SYSTEMS - BYTES


        $$$=.                   ;MARK START OF PLAS AREA
        T.ATT:  .BLKW   2       ;ATTACHMENT DESCRIPTOR LISTHEAD
        T.OFF:  .BLKW   1       ;OFFSET TO TASK IMAGE IN PARTITION
                                ;IF A$$HDR IS DEFINED, THIS WORD ALSO
                                ;INCLUDES THE LENGTH OF THE ALTERNATE
                                ;HEADER REFRESH AREA STORED IN T.HDLN
                .BLKB   1       ;RESERVED
        T.SRCT: .BLKB   1       ;SREF WITH EFN COUNT IN ALL RECEIVE QUEUES
        T.RRFL: .BLKW   2       ;RECEIVE BY REFERENCE LISTHEAD

                .IF NDF P$$LAS
        .=$$$                   ;POINT TO START OF PLAS AREA
                .ENDC   ;P$$LAS
```

```
        .IF NB SYSDEF

$$$=.                          ;MARK START OF PARENT OFFSPRING TASKING AREA
T.OCBH: .BLKW    2             ;OFFSPRING CONTROL BLOCK LISTHEAD
T.RDCT: .BLKW    1             ;OUTSTANDING OFFSPRING COUNT

        .IF NDF P$$OFF
.=$$$                          ;POINT TO START OF PARENT OFFSPRING AREA
        .ENDC   ;P$$OFF


$$$=.                          ;MARK START OF EVENT FLAG MASK AREA
T.EFLM: .BLKW    2             ;EVENT FLAG MASK WORD
                               ;EVENT FLAG MASK ADDRESS

        .IF NDF S$$TOP & T$$BUF
.=$$$                          ;POINT TO START OF EVENT FLAG MASK AREA
        .ENDC   ;S$$TOP & T$$BUF


$$$=.
T.HDLN: .BLKB    1             ;TASK HEADER LENGTH IN 32-WORD BLOCKS

        .IF NDF A$$HDR
.=$$$                          ;NOT SUPPORTED IF NDF
        .ENDC   ;A$$HDR


$$$=.
T.GGF:  .BLKB    1             ;GROUP GLOBAL USE COUNT FOR TASK

        .IF NDF G$$EFN ! R$$SND
.=$$$
        .ENDC   ;G$$EFN ! R$$SND


        .EVEN

T.LGTH=.                       ;LENGTH OF TASK CONTROL BLOCK
T.EXT=0                        ;LENGTH OF TCB EXTENSION

        .IFF

;
; TASK STATUS DEFINITIONS
;
; FIRST STATUS WORD (BLOCKING BITS)
;
TS.EXE=100000                  ;TASK NOT IN EXECUTION (1=YES)
TS.RDN=40000                   ;I/O RUN DOWN IN PROGRESS (1=YES)
TS.MSG=20000                   ;ABORT MESSAGE BEING OUTPUT (1=YES)
TS.NRP=10000                   ;TASK MAPPED TO NONRESIDENT PARTITION (1=YES)
TS.RUN=4000                    ;TASK IS RUNNING ON ANOTHER PROCESSOR (1=YES)
TS.HLD=2000                    ;TASK HALF-LOADED BY TASK LOADER
TS.STP=1000                    ;TASK EXTERNALLY BLOCKED VIA CLI COMMAND
TS.OUT=400                     ;TASK IS OUT OF MEMORY (1=YES)
TS.CKP=200                     ;TASK IS BEING CHECKPOINTED (1=YES)
TS.CKR=100                     ;TASK CHECKPOINT REQUESTED (1=YES)
```

```
;
; TASK BLOCKING STATUS MASK
;
TS.BLK=TS.CKP!TS.CKR!TS.EXE!TS.MSG!TS.NRP!TS.OUT!TS.RDN!TS.STP


;
; SECOND STATUS WORD (STATE BITS)
;
T2.AST=100000           ;AST IN PROGRESS (1=YES)
T2.DST=40000            ;AST RECOGNITION DISABLED (1=YES)
T2.CHK=20000            ;TASK NOT CHECKPOINTABLE (1=YES)
T2.CKD=10000            ;CHECKPOINTING DISABLED (1=YES)
T2.SEF=4000             ;TASK STOPPED FOR EVENT FLAGS (1=YES)
T2.FXD=2000             ;TASK FIXED IN MEMORY (1=YES)
T2.REX=1000             ;ABORT AST EFFECTED OR IN PROGRESS (1=YES)
T2.CAF=400              ;DYN CHECKPOINT SPACE ALLOCATION FAILURE
T2.HLT=200              ;TASK IS BEING HALTED (1=YES)
T2.ABO=100              ;TASK MARKED FOR ABORT (1=YES)
T2.STP=40               ;SAVED T2.STP ON AST IN PROGRESS
T2.STP=20               ;TASK STOPPED (1=YES)
T2.SPN=10               ;SAVED T2.SPN ON AST IN PROGRESS
T2.SPN=4                ;TASK SUSPENDED (1=YES)
T2.WFR=2                ;SAVED T2.WFR ON AST IN PROGRESS
T2.WFR=1                ;TASK IN WAITFOR STATE (1=YES)


;
; THIRD STATUS WORD (ATTRIBUTE BITS)
;
T3.ACP=100000           ;ANCILLARY CONTROL PROCESSOR (1=YES)
T3.PMD=40000            ;DUMP TASK ON SYNCHRONOUS ABORT (0=YES)
T3.REM=20000            ;REMOVE TASK ON EXIT (1=YES)
T3.PRV=10000            ;TASK IS PRIVILEGED (1=YES)
T3.MCR=4000             ;TASK REQUESTED AS EXTERNAL MCR FUNCTION(1=YES)
T3.SLV=2000             ;TASK IS A SLAVE TASK (1=YES)
T3.CLI=1000             ;TASK IS A COMMAND LINE INTERPRETER (1=YES)
T3.RST=400              ;TASK IS RESTRICTED (1=YES)
T3.NSD=200              ;TASK DOES NOT ALLOW SEND DATA
T3.CAL=100              ;TASK HAS CHECKPOINT SPACE IN TASK IMAGE
T3.ROV=40               ;TASK HAS RESIDENT OVERLAYS
T3.NET=20               ;NETWORK PROTOCOL LEVEL
T3.GFL=10               ;TASK HAS ITS GRP GBL EVENT FLAGS LOCKED
;     =4                ;RESERVED FOR FUTURE USE
T3.SWS=2                ;RESERVED FOR USE BY SOFTWARE SERVICES
;     =1                ;RESERVED FOR FUTURE USE

        .ENDC   ;SYSDEF

        .PSECT
```

UCBDF$

```
                 UCBDF$   ,,TTDEF,SYSDEF

        ;
        ; UNIT CONTROL BLOCK
        ;
        ; THE UNIT CONTROL BLOCK (UCB) DEFINES THE STATUS OF AN INDIVIDUAL
        ; DEVICE UNIT AND IS THE CONTROL BLOCK THAT IS POINTED TO BY THE
        ; FIRST WORD OF AN ASSIGNED LUN. THERE IS ONE UCB FOR EACH DEVICE
        ; UNIT OF EACH DCB. THE UCB'S ASSOCIATED WITH A PARTICULAR DCB ARE
        ; CONTIGUOUS IN MEMORY AND ARE POINTED TO BY THE DCB. UCB'S ARE
        ; VARIABLE LENGTH BETWEEN DCB'S BUT ARE OF THE SAME LENGTH FOR A
        ; SPECIFIC DCB. TO FINISH THE TELETYPE EXAMPLE ABOVE, EACH UNIT ON
        ; BOTH INTERFACES WOULD HAVE A UCB.
        ;
                 .ASECT

                 .IF NB   SYSDEF

                 .IF DF   E$$DVC

                 .IF DF   M$$MUP   ;IS U.OWN THERE?
        .=177766
                 .IFF
        .=177770
                 .ENDC    ;M$$MUP

        U.IOC:   .BLKW    2        ;I/O COUNT SINCE MOUNT (ERROR LOG DEVS ONLY)
        U.ERSL:  .BLKB    1        ;SOFT ERROR LIMIT
        U.ERHL:  .BLKB    1        ;HARD ERROR LIMIT
        U.ERSC:  .BLKB    1        ;SOFT ERROR COUNT
        U.ERHC:  .BLKB    1        ;HARD ERROR COUNT

                 .ENDC    ;E$$DVC

                 .ENDC    ;SYSDEF

        .=177772
177772  U.MUP:                     ;MULTIUSER PROTECTION FLAG WORD
177772  U.CLI:   .BLKW    1        ;TCB OF COMMAND LINE INTERPRETER
177774  U.LUIC:  .BLKW    1        ;LOGIN UIC - MULTI USER SYSTEMS ONLY
177776  U.OWN:   .BLKW    1        ;OWNING TERMINAL - MULTI USER SYSTEMS ONLY
000000  U.DCB:   .BLKW    1        ;BACK POINTER TO DCB
000002  U.RED:   .BLKW    1        ;POINTER TO REDIRECT UNIT UCB
000004  U.CTL:   .BLKB    1        ;CONTROL PROCESSING FLAGS
000005  U.STS:   .BLKB    1        ;UNIT STATUS
000006  U.UNIT:  .BLKB    1        ;PHYSICAL UNIT NUMBER
000007  U.ST2:   .BLKB    1        ;UNIT STATUS EXTENSION
000010  U.CW1:   .BLKW    1        ;FIRST DEVICE CHARACTERISTICS WORD
000012  U.CW2:   .BLKW    1        ;SECOND DEVICE CHARACTERISTICS WORD
000014  U.CW3:   .BLKW    1        ;THIRD DEVICE CHARACTERISTICS WORD
000016  U.CW4:   .BLKW    1        ;FOURTH DEVICE CHARACTERISTICS WORD
000020  U.SCB:   .BLKW    1        ;POINTER TO SCB
000022  U.ATT:   .BLKW    1        ;TCB ADDRESS OF ATTACHED TASK
000024  U.BUF:   .BLKW    1        ;RELOCATION BIAS OF CURRENT I/O REQUEST
000026           .BLKW    1        ;BUFFER ADDRESS OF CURRENT I/O REQUEST
000030  U.CNT:   .BLKW    1        ;BYTE COUNT OF CURRENT I/O REQUEST
```

```
000032  U.ACP=U.CNT+2           ;ADDRESS OF TCB OF MOUNTED ACP
000034  U.VCB=U.CNT+4           ;ADDRESS OF VOLUME CONTROL BLOCK
000032  U.CBF=U.CNT+2           ;CONTROL BUFFER RELOCATION AND ADDRESS
000032  U.KCSR=U.CNT+2          ;CSR ADDRESS OF KMC-11
000034  U.KCS6=U.KCSR+2         ;CSR+6 OF KMC-11


        ;
        ;       MAGTAPE DRIVER DEFINITIONS
        ;
000036  U.SPC=U.CNT+6           ;SPACING COUNT
000036  U.SUB=U.CNT+6           ;SUBCONTROLLER, PHYSICAL UNIT #.
000040  U.FNUM=U.CNT+10         ;FORMATTER NUMBER
000042  U.FCDE=U.CNT+12         ;FUNCTION CODE AND INDEX


        ;
        ;       MSCP DISK DRIVER UCB OFFSETS
        ;
000036  U.UTMO=U.VCB+2          ;UNIT COMMAND TIME OUT
000040  U.LHD=U.VCB+4           ;UNIT OUTSTANDING I/O PACKET LISTHEAD
000044  U.BPKT=U.VCB+10         ;UNIT BAD BLOCK PACKET WAITING LIST


        ;
        ; CHARACTERISTICS OBTAINED FROM "GET UNIT STATUS" END PACKETS
        ;
000050  U.MLUN=U.VCB+14         ;MULTI-UNIT CODE
000052  U.UNFL=U.VCB+16         ;UNIT FLAGS
000054  U.HSTI=U.VCB+20         ;HOST IDENTIFIER
000060  U.UNTI=U.VCB+24         ;UNIT IDENTIFIER
000070  U.MEDI=U.VCB+34         ;MEDIA IDENTIFIER
000074  U.SHUN=U.VCB+40         ;SHADOW UNIT
000076  U.SHST=U.VCB+42         ;SHADOW UNIT STATUS
000100  U.TRCK=U.VCB+44         ;UNIT TRACK SIZE
000102  U.GRP=U.VCB+46          ;UNIT GROUP SIZE
000104  U.CYL=U.VCB+50          ;UNIT CYLINDER SIZE
000110  U.RCTS=U.VCB+54         ;UNIT RCT TABLE SIZE
000112  U.RBNS=U.VCB+56         ;UNIT RBN 'S / TRACK
000113  U.RCTC=U.VCB+57         ;UNIT RCT COPIES


        ;
        ; CHARACTERISTICS OBTAINED FROM "ONLINE" OR "SET UNIT CHARACTERISTICS"
        ; END PACKETS
        ;
000114  U.UNSZ=U.VCB+60         ;UNIT SIZE
000120  U.VSER=U.VCB+64         ;VOLUME SERIAL NUMBER


        ;
        ; TERMINAL DRIVER DEFINITIONS
        ;
        .=U.BUF
000024  U.TUX:  .BLKW   1       ;POINTER TO UCB EXTENSION (UCBX)
000026  U.TSTA: .BLKW   3       ;STATUS TRIPLE-WORD
000034  U.TTAB: .BLKW   1       ;IF 0: U.TTAB+1 IS SINGLE-CHARACTER TYPE-AHEAD
                                ;       BUFFER, CURRENTLY EMPTY
                                ;IF ODD: U.TTAB+1 IS SINGLE-CHARACTER
                                ;        TYPE-AHEAD BUFFER AND HOLDS A
                                ;        CHARACTER
                                ;IF NON-0 AND EVEN: POINTER TO MULTI-CHARACTER
                                ;       TYPE-AHEAD BUFFER
```

```
000036  U.TLPP: .BLKB    1          ;LINES PER PAGE
000037  U.TFRQ: .BLKB    1          ;FORK REQUEST BYTE
000040  U.TFLK: .BLKW    1          ;FORK LIST LINK WORD
000042  U.TCHP: .BLKB    1          ;CURRENT HORIZONTAL POSITION
000043  U.TCVP: .BLKB    1          ;CURRENT VERTICAL POSITION
000044  U.UIC:  .BLKW    1          ;TERMINAL UIC
000046  U.TTYP: .BLKB    1          ;TERMINAL TYPE
000047  U.TMTI: .BLKB    1          ;MODEM TIMER
000050  U.CTYP: .BLKW    1          ;CONTROLLER TYPE
000052  U.ACB:  .BLKW    1          ;ANCILLARY CONTROL DRIVER BLOCK ADDR
000054  U.AFLG: .BLKW    1          ;ANCILLARY CONTROL DRIVER FLAGS WORD
000056  U.ADMA: .BLKW    1          ;ANCILLARY CONTROL DRIVER DMA BUFFER


        ;
        ; CONSOLE DRIVER DEFINITIONS
        ;
        .=U.CNT
000030  U.CTCB: .BLKW    1          ;ADDRESS OF CONSOLE LOGGER TCB
000032  U.COTQ: .BLKW    2          ;I/O PACKET LIST QUEUE
000036  U.RED2: .BLKW    1          ;REDIRECT UCB ADDRESS


        ;
        ; DEFINE BITS IN STATUS WORD 1 (U.TSTA)
        ;
        S1.RST=1                    ;READ WITH SPECIAL TERMINATORS IN PROGRESS
        S1.RUB=2                    ;RUBOUT SEQUENCE IN PROGRESS (NON-SCOPE)
        S1.ESC=4                    ;ESCAPE SEQUENCE IN PROGRESS
        S1.RAL=10                   ;READ ALL IN PROGRESS
        S1.RNE=20                   ;ECHO SUPPRESSED
        S1.CTO=40                   ;OUTPUT STOPPED BY CTRL-O
        S1.OBY=100                  ;OUTPUT BUSY
        S1.IBY=200                  ;INPUT BUSY
        S1.BEL=400                  ;BELL PENDING
        S1.DPR=1000                 ;DEFER PROCESSING OF CHAR. IN U.TECB
        S1.DEC=2000                 ;DEFER ECHO OF CHAR. IN U.TECB
        S1.DSI=4000                 ;INPUT PROCESSING DISABLED
        S1.CTS=10000                ;OUTPUT STOPPED BY CTRL-S
        S1.USI=20000                ;UNSOLICITED INPUT IN PROGRESS
                                    ;BIT 14 RESERVED FOR NON-BUFFERED OUTPUT
        S1.OBF=40000                ;BUFFERED OUTPUT IN PROGRESS
        S1.IBF=100000               ;BUFFERED INPUT IN PROGRESS


        ;
        ; DEFINE BITS IN STATUS WORD 2 (U.TSTA+2)
        ;
        S2.ACR=1                    ;WRAP-AROUND (AUTOMATIC CR-LF) REQUIRED
        S2.WRA=6                    ;CONTEXT FOR WRAP-AROUND
        S2.WRB=2                    ;LOW BIT IN S2.WRA BIT PATTERN
        S2.CR=10                    ;TRAILING CR REQUIRED ON OUTPUT
        S2.BRQ=20                   ;BREAK-THROUGH-WRITE REQUEST IN QUEUE
        S2.SRQ=40                   ;SPECIAL REQUEST IN QUEUE
                                    ;(IO.ATT, IO.DET, SF.SMC)
        S2.ORQ=100                  ;OUTPUT REQUEST IN QUEUE (MUST = S1.OBY)
        S2.IRQ=200                  ;INPUT REQUEST IN QUEUE (MUST = S1.IBY)
        S2.HFL=3400                 ;HORIZONTAL FILL REQUIREMENT
        S2.VFL=4000                 ;VERTICAL FILL REQUIREMENT
        S2.HHT=10000                ;HARDWARE HORIZONTAL TAB PRESENT
        S2.HFF=20000                ;HARDWARE FORM-FEED PRESENT
        S2.FLF=40000                ;FORCE LINE FEED BEFORE NEXT ECHO
        S2.FDX=100000               ;LINE IS IN FULL DUPLEX MODE
```

```
;
; DEFINE BITS IN STATUS WORD 3 (U.TSTA+4)
;
S3.RAL=10                  ;TERMINAL IS IN READ-PASS-ALL MODE
                           ;(S3.RAL MUST = S1.RAL)
S3.RPO=20                  ;READ W/PROMPT OUTPUT IN PROGRESS
S3.WES=40                  ;TASK WANTS ESCAPE SEQUENCES
S3.TAB=100                 ;TYPE-AHEAD BUFFER ALLOCATION REQUESTED
S3.8BC=200                 ;PASS 8 BITS ON INPUT
S3.RCU=400                 ;RESTORE CURSOR (MUST = TF.RCU*400)
S3.ABD=1000                ;AUTO-BAUD SPEED DETECTION ENABLED
S3.ABP=2000                ;AUTO-BAUD SPEED DETECTION IN PROGRESS
S3.WAL=4000                ;WRITE-PASS-ALL (MUST = TF.WAL*400)
S3.VER=10000               ;LAST CHAR. IN TYPE-AHEAD BUFFER
                           ;HAS PARITY ERROR
S3.BCC=20000               ;LAST CHAR. IN TYPE-AHEAD BUFFER
                           ;HAS FRAMING ERROR
S3.DAO=40000               ;LAST CHAR. IN TYPE-AHEAD BUFFER
                           ;HAS DATA OVERRUN ERROR
                           ;NOTE - THE 3 BITS ABOVE MUST CORRESPOND
                           ;TO THE RESPECTIVE ERROR FLAGS IN THE
                           ;HARDWARE RECEIVE BUFFER
S3.PCU=100000              ;POSITION CURSOR (MUST = TF.PCU*400)

        .PSECT

;
; DEVICE TABLE STATUS DEFINITIONS
;
; DEVICE CHARACTERISTICS WORD 1 (U.CW1) DEVICE TYPE DEFINITION BITS.
;
DV.REC=1                   ;RECORD ORIENTED DEVICE (1=YES)
DV.CCL=2                   ;CARRIAGE CONTROL DEVICE (1=YES)
DV.TTY=4                   ;TERMINAL DEVICE (1=YES)
DV.DIR=10                  ;FILE STRUCTURED DEVICE (1=YES)
DV.SDI=20                  ;SINGLE DIRECTORY DEVICE (1=YES)
DV.SQD=40                  ;SEQUENTIAL DEVICE (1=YES)
DV.MSD=100                 ;MASS STORAGE DEVICE (1=YES)
DV.UMD=200                 ;USER MODE DIAGNOSTICS SUPPORTED (1=YES)
DV.MBC=400                 ;DEVICE IS ON MASSBUS CONTROLLER (1=YES)
DV.EXT=400                 ;DEVICE ON EXTENDED ADDRESSING CONTROLLER
DV.SWL=1000                ;UNIT SOFTWARE WRITE LOCKED (1=YES)
DV.ISP=2000                ;INPUT SPOOLED DEVICE (1=YES)
DV.OSP=4000                ;OUTPUT SPOOLED DEVICE (1=YES)
DV.PSE=10000               ;PSEUDO DEVICE (1=YES)
DV.COM=20000               ;DEVICE IS MOUNTABLE AS COM CHANNEL (1=YES)
DV.F11=40000               ;DEVICE IS MOUNTABLE AS F11 DEVICE (1=YES)
DV.MNT=100000              ;DEVICE IS MOUNTABLE (1=YES)


;
; TERMINAL DEPENDENT CHARACTERISTICS WORD 2 (U.CW2) BIT DEFINITIONS
;
U2.DH1=100000              ;UNIT IS A MULTIPLEXER (1=YES)
U2.DJ1=40000               ;UNIT IS A DJ11 (1=YES)
U2.RMT=20000               ;UNIT IS REMOTE (1=YES)
U2.HFF=10000               ;UNIT HANDLES HARDWARE FORM FEEDS (1=YES)
U2.L8S=10000               ;OLD NAME FOR U2.HFF
U2.NEC=4000                ;DON'T ECHO SOLICITED INPUT (1=YES)
U2.CRT=2000                ;UNIT IS A CRT (1=YES)
U2.ESC=1000                ;UNIT GENERATES ESCAPE SEQUENCES (1=YES)
U2.LOG=400                 ;USER LOGGED ON TERMINAL (0=YES)
U2.SLV=200                 ;UNIT IS A SLAVE TERMINAL (1=YES)
U2.DZ1=100                 ;UNIT IS A DZ11 (1=YES)
```

```
U2.HLD=40               ;TERMINAL IS IN HOLD SCREEN MODE (1=YES)
U2.AT.=20               ;MCR COMMAND AT. BEING PROCESSED (1=YES)
U2.PRV=10               ;UNIT IS A PRIVILEGED TERMINAL (1=YES)
U2.L3S=4                ;UNIT IS A LA30S TERMINAL (1=YES)
U2.SCS=4                ;SCS-11 COMMAND TERMINAL (1=YES)
U2.VT5=2                ;UNIT IS A VT05B TERMINAL (1=YES)
U2.LWC=1                ;LOWER CASE TO UPPER CASE CONVERSION (0=YES)


;
; BIT DEFINITIONS FOR U.MUP (SYSTEMS WITH ALTERNATE CLI SUPPORT ONLY)
;
UM.OVR=1                ;OVERRIDE CLI INDICATOR
UM.CLI=36               ;CLI INDICATOR BITS
UM.DSB=200              ;TERMINAL DISABLED SINCE CLI ELIMINATED
UM.NBR=400              ;NO BROADCAST


;
; RH11-RS03/RS04 CHARACTERISTICS WORD 2 (U.CW2) BIT DEFINITIONS
;
U2.R04=100000           ;UNIT IS A RS04 (1=YES)


;
; RH11-TU16 CHARACTERISTICS WORD 2 (U.CW2) BIT DEFINITIONS
;
U2.7CH=10000            ;UNIT IS A 7 CHANNEL DRIVE (1=YES)


;
; TERMINAL DEPENDENT CHARACTERISTICS WORD 3 (U.CW3) BIT DEFINITIONS
;
U3.UPC=20000            ;UPCASE OUTPUT FLAG


;
; UNIT CONTROL PROCESSING FLAG DEFINITIONS
;
UC.ALG=200              ;BYTE ALIGNMENT ALLOWED (1=NO)
UC.NPR=100              ;DEVICE IS AN NPR DEVICE (1=YES)
UC.QUE=40               ;CALL DRIVER BEFORE QUEUING (1=YES)
UC.PWF=20               ;CALL DRIVER AT POWERFAIL ALWAYS (1=YES)
UC.ATT=10               ;CALL DRIVER ON ATTACH/DETACH (1=YES)
UC.KIL=4                ;CALL DRIVER AT I/O KILL ALWAYS (1=YES)
UC.LGH=3                ;TRANSFER LENGTH MASK BITS


;
; UNIT STATUS BIT DEFINITIONS
;
US.BSY=200              ;UNIT IS BUSY (1=YES)
US.MNT=100              ;UNIT IS MOUNTED (0=YES)
US.FOR=40               ;UNIT IS MOUNTED AS FOREIGN VOLUME (1=YES)
US.MDM=20               ;UNIT IS MARKED FOR DISMOUNT (1=YES)
US.PWF=10               ;POWERFAIL OCCURRED (1=YES)


;
; CARD READER DEPENDENT UNIT STATUS BIT DEFINITIONS
;
US.ABO=1                ;UNIT IS MARKED FOR ABORT IF NOT READY (1=YES)
US.MDE=2                ;UNIT IS IN 029 TRANSLATION NODE (1=YES)
```

```
;
; FILES-11 DEPENDENT UNIT STATUS BITS
;
US.WCK=10               ;WRITE CHECK ENABLED (1=YES)
US.SPU=2                ;UNIT IS SPINNING UP (1=YES)
US.VV=1                 ;VOLUME VALID IS SET (1=YES)


;
; KMC-11-LP DEPDENDENT UNIT STATUS BITS
;
US.KPF=1                ;KMC-11 POWERFAIL INTERLOCK


;
; TERMINAL DEPENDENT UNIT STATUS BIT DEFINITIONS
;
        .IF NB  TTDEF

        .IF DF T$$CPW

US.CRW=4                ;UNIT IS WAITING FOR CARRIER (1=YES)
US.DSB=2                ;UNIT IS DISABLED (1=YES)
US.OIU=1                ;OUTPUT INTERRUPT IS UNEXPECTED ON UNIT (1=YES)

        .IFF    ;T$$CPW

US.DSB=10               ;UNIT IS DISABLED (1=YES)
US.CRW=4                ;UNIT IS WAITING FOR CARRIER (1=YES)
US.ECH=2                ;UNIT HAS ECHO IN PROGRESS (1=YES)
US.OUT=1                ;UNIT IS EXPECTING OUTPUT INTERRUPT (1=YES)

        .ENDC   ;T$$CPW

        .ENDC   ;TTDEF

;
; LPS11 DEPENDENT UNIT STATUS BIT DEFINITIONS
;
US.FRK=2                ;FORK IN PROGRESS (1=YES)
US.SHR=1                ;SHAREABLE FUNCTION IN PROGRESS (0=YES)


;
; MAGTAPE DEPENDANT UNIT STATUS BITS
;
US.LAB=4                ;UNIT HAS LABELED TAPE ON IT (1=YES)
US.BSP=2                ;INTERNAL BACKSPACE IN PROGRESS (1=YES)
```

```
;
; UNIT STATUS EXTENSION (U.ST2) BIT DEFINITIONS
;
US.OFL=1                ;UNIT OFFLINE (1=YES)
US.RED=2                ;UNIT REDIRECTABLE (0=YES)
US.PUB=4                ;UNIT IS PUBLIC DEVICE (1=YES)
US.UMD=10               ;UNIT ATTACHED FOR DIAGNOSTICS (1=YES)


;
; MAG TAPE DENS SUPPORT IDENT IN CHAR WORD 3 (U.CW3) DEFENITION
;       ASSIGNMENTS PER NUMERICAL SEQUENCE 0 - 255.
;
UD.UNS=0                ;UNSUPPORTED
UD.200=1                ; 200BPI, 7 TRACK
UD.556=2                ; 556BPI, 7 TRACK
UD.800=3                ; 800BPI, 7 OR 9 TRACK
UD.160=4                ;1600BPI, 9 TRACK
UD.625=5                ;6250BPI, 9 TRACK
```

APPENDIX D

USER-WRITTEN ANCILLARY CONTROL PROCESSORS

This chapter is intended as a guide for the user in developing an Ancillary Control Processor (ACP). It is not a tutorial and it is not a description of the logic of any DIGITAL-supplied ACP. You should be thoroughly familiar with the RSX-11M Guide to Writing an I/O Driver.

This chapter provides the following information:

- An overview of the RSX-11M I/O system
- Descriptions of the types of ACPs
- The attributes of an ACP
- A description of the flow of an input/output request, emphasizing the role of the ACP
- System data structures used by the ACPs
- Examples of an ACP and an I/O driver

D.1 OVERVIEW OF THE RSX-11M I/O SYSTEM

An Ancillary Control Processor (ACP) is one component of the RSX I/O system. The other major components are

- File Control Services (FCS) or Record Management Services (RMS)
- QIO$ directive processing in the Executive
- Device drivers

Figure D-1 shows how an ACP fits into the overall structure.

The philosophy and structure of the RSX-11M I/O system are described in detail in Chapter 2 of this manual. QIO$ directive processing is described in the RSX-11M/M-PLUS Executive Reference Manual.

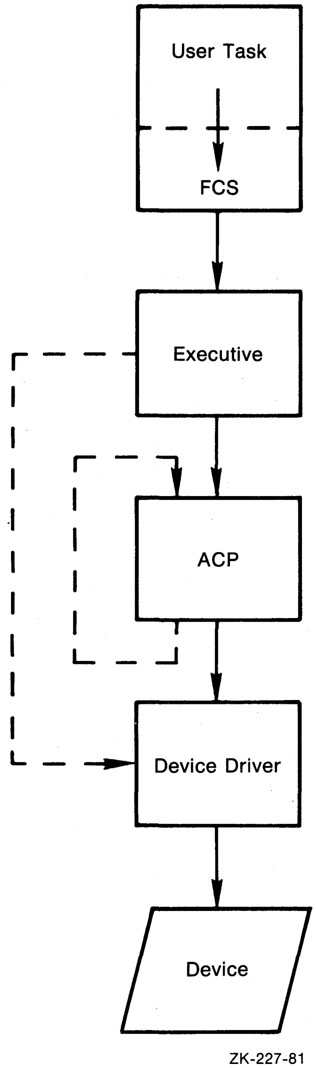


Figure D-1 The RSX-11M I/O System

D.2 TYPES OF ANCILLARY CONTROL PROCESSORS

An Ancillary Control processor (ACP) is a task that provides extended functions (complex operations requiring either multiple I/O requests or special privileged operations) for a class of I/O devices.

ACPs may be divided into three types:

- Those that manage file structures, such as F11ACP and MTAACP
- Those that manage intertask or interprocessor communication, such as the network ACP (NETACP)
- Those that perform special privileged operations on behalf of nonprivileged user tasks.

Some of the purposes of a user-written ACP are:

- To implement a foreign file system
- To extend the capabilities of a DIGITAL-supplied device driver
- To extend the services of the operating system
- To implement a communications protocol

User-written ACPs extend functionality, not performance. If your application is performance-oriented, you should consider writing a special driver rather than an ACP.

D.2.1 ACPs Which Manage Files Structures

DIGITAL supplies F11ACP for Files-11 disk structure and MTAACP for ANSI magnetic tapes. You may write an ACP to implement a foreign file system or tape format. Changes to the Executive, the I/O driver, or the data structures are not necessary if there are DIGITAL-supplied I/O operations that correspond to operations for the foreign format. The user-written ACP can use the built-in Executive Services (such as QIO$ directive processing) without change.

Note that a user-written ACP is necessary only to support a file structure other than Files-11. To use the Files-11 structure with a foreign device, you need to write a device driver, and you may need to modify or extend disk initialization, management, or backup utilities.

D.2.2 ACPs Which Manage Intertask or Interprocessor Communication

DECnet/M and DECnet/M-PLUS both contain an example of an ACP that manages interprocessor communications: NETACP, used for management of the Digital Network Architecture Communications protocol. You may write an ACP to manage a foreign communications protocol.

D.2.3 ACPs Which Perform Privileged Operations for Unprivileged Tasks

You may write an ACP to support extended capabilities for a class of devices (such as line printer spooling or associative name searching for data base devices). This requires special support in the Executive I/O processing, in the I/O driver, and in the associated data structures. This type of ACP requires a user-written I/O driver that contains special code to support the ACP.

An ACP may use the I/O driver interface to extend the services of the operating system (as in the case of a data base management system), rather than to extend the services of an I/O device.

RSX-11M contains no DIGITAL-supplied examples of the types just described. Both RSX-11M and RSX-11M-PLUS, however, contain COT and CODRV, which are used to perform console logging. The COT task behaves in the fashion of an ACP, in that it receives I/O packets in its queue from user tasks. COT is not declared to the system as an ACP, though; it has no VCB or MOU interface. COT is enabled and disabled using MCR commands.

D.3 THE ATTRIBUTES OF AN ACP

The classes of ACPs described above have the following attributes in common, which further define an ACP:

- An ACP is an asynchronous privileged task. Refer to the RSX-11M/M-PLUS Task Builder Manual for a discussion of privileged task mapping and Executive access.
- An ACP implements a protocol (or set of services) for a class of devices (for example, file-structured devices or sequential devices).
- An ACP functions as an extension of the Executive and frequently operates with Executive privilege.
- An ACP can be enabled or disabled for a particular device.
- An ACP is shareable among several device drivers and units.

D.3.1 ACP as a Task

An ACP is a task. It has all the attributes of a task, including:

- A stack
- A task name
- Priority
- Scheduling by the Executive

An ACP, in contrast to a driver, operates as a task. While a driver handles interrupts, manipulates CSRs, and performs other device-specific operations, an ACP is not device bound and can take advantage of the services and control mechanisms available to tasks. ACPs can also use overlays, and can therefore be larger and have greater functionality than drivers.

Because the ACP task is privileged, it has access to the Executive data structures and can use Executive facilities.

Unlike other privileged tasks, an ACP has the capacity to receive I/O packets from other tasks by means of the QIO$ directive. This permits the ACP to act as an I/O handler, which can compete with user tasks for system resources more equitably than an I/O driver could. Also, unlike an I/O driver, an ACP can perform I/O to other devices during the processing of an I/O request.

D.3.2 Class of Devices

An ACP can easily implement functions for a class of devices because it communicates via the QIO$ directive, which is relatively device-independent. (Drivers, in contrast, are written to suit the minute peculiarities of particular devices.) F11ACP, for example, implements the Files-11 structure for all types of disks and DECtapes. MTAACP implements ANSI magnetic tape format for all DIGITAL-supported 9-track magnetic tape drives.

D.3.3 Extension of Executive

An ACP extends the functionality of both the Executive and the device drivers in several ways:

- By removing the burden of device managment (assigning space on a volume, locating the desired data area, and so on) from the programmer.
- By handling the manipulation of device- and protocol-related data.
- By permitting a device to be treated as a logical rather than a physical entity.
- By allowing a device to be shared for simultaneous access. Each process that accesses a device is protected from other processes by the ACP and its protocol. The ACP also synchronizes access to the physical device.

D.3.4 Enabling Capability and Disabling Capability

An ACP can be enabled or disabled for a given device. When enabled, the device is available for use in the context of the protocol provided by the ACP.

D.3.5 Shareability

An ACP is shareable among several device drivers and units.

D.4 THE FLOW OF AN I/O REQUEST

This section describes the system interactions of an ACP during the flow of an I/O request. On the assumption that you have read Section 2.6 of this manual, only those items relevant to ACPs are treated in detail. The structure of this section is similar to that of Section 2.6.

The I/O flow proceeds as described below:

1. [Task issues a QIO$ directive]

   The Executive performs the following:

   a. First-level validity checks

   b. Redirect algorithm

   c. Additional validity checks

2. [Executive obtains storage for and creates I/O packet]

3. [Executive validates the function requested]

   If the function is an ACP function, the type of function validation depends on the type of ACP.

a. Standard DIGITAL-supplied ACP

If the device is mounted with an ACP, the function code is validated to determine that it is an ACP function. The parameters are verified by code in the QIO processing module. The request is checked for proper order (for example, OPEN before CLOSE) and for valid buffers. The task that issues the I/O request must validate any additional parameter block required by the ACP. For some functions, an additional parameter buffer is allocated and filled in.

Sometimes the I/O request can be transformed into a transfer function and queued to the proper driver. If the request cannot be queued to the driver or cannot be completed immediately, it is queued to the ACP. The request packet is inserted in the ACP's receive data queue and the ACP is unstopped or requested to run (depending on whether it was active).

b. Nonstandard ACPs and ACPs requiring special Executive or driver support

The QIO$ directive processing cannot validate the ACP function request parameters. The I/O driver must do the validation. The UC.QUE bit in the driver must be set so that the QIO processing routine calls the driver directly, without queuing the I/O packet to the driver or to the ACP.

The driver must perform the same functions for the ACP as the QIO processing code does for standard and replacement ACPs.

4. [Driver processing]

If the driver calls $GTPKT and the next request in the queue is an ACP function request, $GTPKT will queue the request to the ACP and activate the ACP.

5. [ACP processing]

Obtain Work

As soon as the ACP is activated, it attempts to remove an I/O request packet from its receive data queue. To do this, the ACP switches to system state and calls $QRMVF to obtain the address of the I/O packet. (Note that the ACP does not use the RCVD$ directive to remove the packet from its receive queue.) When the ACP has obtained the address of the I/O packet, it returns to task state.

Process I/O Request

The ACP either completes the I/O request or translates it into a standard I/O driver request. The decision is based on:

- Information in the data structures
- Computation
- Additional I/O

The ACP may do its own QIO requests to the driver; the ACP then calls $IOFIN with the I/O packet and the I/O status. Alternatively, when the I/O request is translated into a driver request, the ACP modifies the I/O packet to put it into the correct form for a driver request and passes it to the driver by calling $DRQRQ. If the I/O request has been completed, the ACP calls $IOFIN with the I/O packet and the I/O status.

For example, MTAACP always deals with the driver through QIO requests, and finishes I/O itself by calling $IOFIN.

The ACP then attempts to remove another request from its queue. If there are no more requests in the queue, the ACP stops itself by calling $STPCT.

D.5 SYSTEM DATA STRUCTURES

An ACP interfaces to the system through use of system data structures and through calls to Executive routines. This section describes ACP-specific information for the various data structures. Detailed information on the data structures and their interrelationships is contained in Section 2.7 and Chapter 4 of this manual.

The following data structures comprise the complete set for I/O processing:

- Task header
- Window Block (WB)
- File Control Block (FCB)
- $DEVHD word, the Device Control Block (DCB), and the Driver Dispatch Table (DDT)
- Unit Control Block (UCB)
- Status Control Block (SCB)
- Volume Control Block (VCB)
- I/O packet

When you write an ACP, you are usually concerned only with the I/O packet, the DCB, the UCB, and the SCB. There are ACP-specific additions to and variations in the I/O packet, the DCB, and the UCB. The SCB is the same as that for an I/O driver.

D.5.1 The I/O Packet

Figure D-2 shows the layout of the 18-word I/O packet, which is constructed by QIO$ directive processing. The fields in the I/O packet are described in detail in Section 4.1.1 of this manual. The following additions and variations exist for ACPs:

I.FCN

Contains the function code for the I/O request. The function code and modifier comprise a 16-bit field. Bits 13,14, and 15 are reserved for ACP interface use. You must not assume that these bits are zero.

If the function code field is referenced, the value should be masked with 160000(8) for example:

```
MOV     I.FCN(R1),R0      ; GET FUNCTION CODE FIELD
BIC     #160000,R0        ; CLEAR OFF EXTRANEOUS BITS
```

Although these bits are not currently in use, they should not be assumed to be zero in order to ensure future compatibility. If an I/O packet is requeued to the device driver, these bits must be cleared.

I.PRM

Contains the device-dependent parameters constructed from the last six words in the DPB.

The following fields are defined for Files-11 nontransfer operations:

```
.ASECT
.=I.PRM
I.FIDP: .BLKW   2         ; File ID address (Address double
                          ; word format)
I.RWAT: .BLKW   1         ; Attribute block pointer (This field
                          ; points to a buffer containing the
                          ; attribute list and associated address
                          ; doublewords)
I.EXTD: .BLKW   2         ; Extend control from parameter list
I.RTRV: .BLKB   1         ; Retrieval pointers desired
I.ACTL: .BLKB   1         ; Access control
I.FNBP: .BLKW   2         ; File name block pointer
```

The following fields are defined for Files-11 transfer operations:

```
.ASECT
.=I.PRM
I.RWAD: .BLKW   2         ; Transfer address doubleword
I.RWCT: .BLKW   1         ; Transfer count
        .BLKW   1         ; Unused
I.RWVB  .BLKW   1         ; Virtual block number
        .BLKW   1         ; Unused
I.LCKB: .BLKW   1         ; Address of lock block
```

The above fields are set up by routines in the Executive.

Only I.LCKB is referenced by $IOFIN; it must be either 0 or greater than 140000.

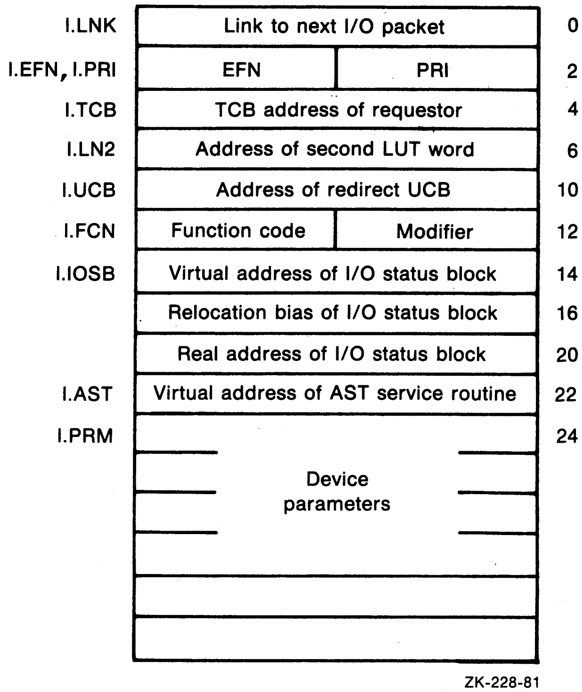


Figure D-2 I/O Packet

D.5.2 The DCB

The DCB is described in detail in Section 4.1.2 of this manual. The following additional information applies to ACPs:

D.MSK

In general, I/O requests marked as ACP functions are passed to the ACP; all others are passed to the device driver when it calls $GTPKT.

D.5.3 The UCB

The UCB is described in detail in Section 4.1.4 of this manual. The following additional information applies to ACPs:

U.CTL

UC.QUE - Queue bypass bit

If UC.QUE is set, all I/O requests are passed to the driver without being queued first, regardless of any ACP-related processing in DRQIO. A driver that makes use of this option may alter the behavior of the file system, since some functions are normally passed directly to the ACP, bypassing the driver queue.

If UC.QUE and US.FOR are set, all DIGITAL-specific ACP processing is bypassed. In this case, the driver must do all address checking and relocation.

The UC.QUE option allows user-written ACPs to validate the ACP function parameters. Each I/O driver supported by the user-written ACP must contain code to do the validation. The validation must be done at this point in the I/O processing, because the routines that do address checking and relocation assume that the memory management registers are in use by the tasks issuing the I/O. (Refer to the DRQIO module for examples of the techniques used to validate and save parameters.)

UC.KIL - Unconditional cancel I/O call bit

If UC.KIL is set, the I/O driver is called on a cancel I/O request even if the unit specified is not busy.

Since ACP functions are dependent on sequencing, this function is normally turned into a no-op.

An IO.KIL request has no effect on a mounted unit since the I/O queue is not flushed on the IO.KIL request when the DV.MNT (mountable) and US.MNT (mounted) bits are set for the unit.

U.STS

US.MNT

If US.MNT is set, the unit is not mounted. US.MNT is cleared when the volume is mounted.

US.FOR

If US.FOR is set, the volume is mounted as foreign. US.FOR is set when the volume is mounted.

US.MDM

If US.MDM is set, the volume is marked for dismount.

An ACP normally checks the US.MDM bit when it processes a new I/O request. The ACP refuses operations that create a channel for processing (such as OPEN) when US.MDM is set. After operations that terminate a channel (such as CLOSE), the ACP checks the current count of active channels to see if all ACP-related processing has been completed. If it is, the ACP completes the dismount.

US.MDM is set when the volume is to be dismounted.

U.CW1

This characteristics word is returned to the user task by the GLUN$ directive. U.CW1 is checked during validation of I/O request. The following bits are important to ACPs:

DV.MNT

If DV.MNT is set, the device is mountable. If a device is mountable, ACP functions can only be performed when it is mounted.

DV.F11

If DV.F11 is set, the device is a Files-11 device. DV.F11 is set for both disks and tapes. If the device is Files-11, the DV.SQD bit determines whether the device is random or sequential.

DV.COM

If DV.COM is set, the device is mountable as a communications channel. This is used for DECnet.

DV.SQD

If DV.SQD is set, the device is sequential. If DV.SQD is reset, the device is random access.

DV.SDI

If DV.SDI is set, the device supports only a single directory.

DV.DIR

If DV.DIR is set, the device is a directory device.

D.6 AN EXAMPLE OF AN ACP-I/O DRIVER COMBINATION

The following is an example of an ACP, including a special driver used by the ACP. This ACP, supplied for demonstration purposes only, counts the number of QIOs to a terminal.

The modules supplied and their respective functions are:

QDPRE.MAC - Prefix file for assembly
QDDAT.MAC - Driver database
QDDRV.MAC - Driver for ACP
QDACP.MAC - The ACP itself
QDCON.MAC - The task for enabling and disabling the ACP

Example D-1 An ACP-I/O Driver Combination

```
        .TITLE  QDPRE
        .IDENT  /01/

        .ENABL  LC

;
; This structure is purely for purposes of example.  It is not intended to
; be useful nor is it supported in any way.  It is, however,
; functional, complete, and representative of a valid interface.
;
```

```
;
; **-QDPRE-QD: driver prefix file
;
;
; EXECUTIVE DEPENDENCIES
;
; The following is a list of the recognized Executive dependencies for the
; the QD: driver.  If the implementation or functionality of the following
; features change, this driver and ACP may not function properly.
;
;    1. The Executive I/O processing as described in RSX-11M Guide to Writing an I/O
;       Driver remains unchanged.
;
;    2. The following Executive routines remain unchanged:
;               $SWSTK   (SWSTK$)
;               $BLXIO
;               $EXRQP
;       All of the routines documented in the RSX-11M Guide to Writing an I/O Driver
;


;
; System Macro Calls
;

        .MCALL  UCBDF$
        UCBDF$


;
; QD driver-specific offsets
;

        .ASECT
.=U.VCB+2                       ; End of UCB

U.QACP::.BLKW   1               ; Address of QD ACP TCB
U.QCTL::.BLKW   1               ; ACP control and status word
U.QLUN::.BLKW   1               ; LUN used by ACP when doing I/O for this unit
U.QTRN::.BLKW   1               ; Count of transactions outstanding to ACP

UQ.STP==100000                  ; Stop requested for ACP on this unit
UQ.ONL==40000                   ; This unit onlne

        .PSECT

Q$$D11=3                        ; Number of units
LD$QD=0                         ; Driver loadable



        .TITLE  QDDAT
        .IDENT  /01/

        .ENABL  LC
```

```
;
; This structure is purely for purposes of example.  It is not intended to
; be useful nor is it supported in any way.  It is, however,
; functional, complete, and representative of a valid interface.
;


;
; **-QDDAT-QD: driver device tables
;
;
; The following data structure is designed with two things in mind:
;       1. Providing the minimum structures to look like a disk
;       2. Providing the minimum structures to satisfy the Executive
;          and the MCR LOA commands.
;

$QDDAT::                                ; Start of QDDRV device tables

QDDCB:  .WORD   0                       ; Link to next DCB
        .WORD   .QD0                    ; Pointer to first UCB
        .ASCII  /QD/                    ; Device name
        .BYTE   0,Q$$D11-1              ; Lowest and highest unit number
        .WORD   QDND-QDST               ; Length of UCB
        .WORD   0                       ; Pointer to driver dispatch table, set by LOA

;
; The following table defines the initial processing of I/O functions in the
; Executive QIO directive processing.  The legal functions selected are those
; of the standard disk drivers.
;

        .WORD   177037                  ; Legal functions 0.-15.
        .WORD   000030                  ; Control functions 0.-15.
        .WORD   000000                  ; No-op functions 0.-15.
        .WORD   177000                  ; ACP functions 0.-15.
        .WORD   000777                  ; Legal functions 16.-31.
        .WORD   000000                  ; Control functions 16.-31.
        .WORD   000000                  ; No-op functions 16.-31.
        .WORD   000777                  ; ACP functions 16.-31.
        .WORD   0                       ; PCB address of driver partition

$$$=0

        .NLIST  MD
        .LIST   ME

        .REPT   Q$$D11

        .IRP    XX,\<$$$>

QDST=.                                  ; Start of UCB

        .IF DF  M$$MUP

        .WORD   0                       ; Login UIC, multi-user protection system
        .WORD   0                       ; Owning terminal UCB address

        .ENDC
```

```
.QD'XX':.WORD   QDDCB             ; Back pointer to DCB
        .WORD   .QD'XX            ; Redirect pointer
        .BYTE   UC.PWF!UC.ALG!3   ; Control flags byte, call on powerfail
                                  ; to allow proper setting of on-line/off-line bit
        .BYTE   US.MNT            ; Status byte
        .BYTE   0                 ; Physical unit number, does not apply
        .BYTE   0                 ; Second status word
        .WORD   DV.MNT!DV.F11!DV.SDI!DV.DIR ; Characteristic word 1
        .WORD   0                 ; Characteristic word 2, size of device
        .WORD   0                 ; Characteristic word 3
        .WORD   512.              ; Characteristic word 4, buffer size
        .WORD   $QD0              ; Pointer to SCB
        .WORD   0                 ; Attached task TCB address
        .BLKW   2                 ; User buffer pointer
        .BLKW   1                 ; and byte count
        .WORD   0                 ; Address of file system ACP TCB
        .WORD   0                 ; Address of VCB for file system
        .WORD   0                 ; U.QACP-QDACP TCB address
        .WORD   0                 ; U.QCTL-QDACP Control word
        .WORD   'XX'+1            ; U.QLUN-QDACP LUN for I/O

QDND=.                            ; End of UCB

        .ENDR

$$$=$$$+1

        .ENDR

        .NLIST  ME
        .LIST   MD

$QD0::  .WORD   0,.-2             ; Device I/O queue
        .BYTE   0,0               ; Device priority and vector
        .BYTE   0,0               ; Current and initial device timeout count
        .BYTE   0,0               ; Controller index and device status
        .WORD   0                 ; CSR address
        .WORD   0                 ; Address of I/O packet
        .BLKW   5                 ; Fork block

$QDEND::                          ; End of QDDRV device tables

        .END



        .TITLE  QDDRV
        .IDENT  /01/

        .ENABL  LC

;
; This driver is purely for purposes of example.  It is not intended to
; be a useful driver nor is it supported in any way.  It is, however,
; functional, complete, and representative of a valid interface.
;

;
; **-QDDRV-QD: driver
;
;
```

```
;
; MACRO LIBRARY CALLS
;

;
; DRIVER DISPATCH TABLE
;

$QDTBL::.WORD   QDINI               ; Initiator entry point
        .WORD   QDCAN               ; Cancel I/O entry point
        .WORD   QDOUT               ; Device timeout entry point
        .WORD   QDPWF               ; Powerfail entry point

;
; LOCAL DATA
;

        .IF DF  AUTOST

ACPTNM: .RAD50  /QDACP /            ; Default ACP task name

        .ENDC

;
; **-QDINI - "Disk" Driver
;
; This driver, in conjunction with its Ancillary Control Processor (ACP)
; appears to be a disk, but its operational characteristics are
; unusual.  The actual storage medium may be any of a number of devices
; including memory, disk, or DECnet link.  No driver queue is maintained;
; all I/O packets are queued directly to the ACP.  The cancel I/O, device
; timeout, and powerfail entry points are all set to be no-ops.  The actual
; processing of the request is left to the ACP.
;
; Remember, this driver is an example and demonstrates multiple features
; of the driver/ACP/Executive interface.
;
; Inputs:
;       R5=UCB address
;
; The buffer address and count for IO.RLB and IO.WLB have been validated
; by DRQIO processing code.  IO.KIL, IO.ATT, and IO.DET are processed by
; the Executive's standard I/O processing routines.  IO.CTL is queued directly
; to QDACP.
;

        .ENABL  LSB

QDINI:                              ; Driver initiation entry point

        CALL    $GTPKT              ; Get I/O packet
        BCS     EXIT                ; If CS none to get
        CLRB    S.STS(R4)           ; Unbusy controller since ACP does all work
        MOV     R1,R3               ; Copy I/O packet address
        BIT     #UQ.ONL,U.QCTL(R5) ; Unit online?
        BEQ     20$                 ; If EQ no
        MOVB    I.FCN+1(R3),R0      ; Get function code
        CMPB    #IO.RLB/400,R0      ; Read logical block?
        BEQ     QPRLB               ; If EQ yes
        CMPB    #IO.WLB/400,R0      ; Write logical block?
        BNE     ERRIFC              ; If NE no
```

```
;
; Function specific validation routines.  The checks here could
; be made later in the ACP, but they are easily made here and have
; the added benefit of saving of two context switches (to and from
; the ACP) to return an error.
;

QPWLB:  MOV     #IE.WLK&377,R0    ; Assume write locked
        BIT     #DV.SWL,U.CW1(R5) ; Unit software write locked?
        BNE     IOFIN             ; If NE yes
                                  ; Join common code

QPRLB:  CALL    $BLKCK            ; Check for valid logical blocks

        MOV     R3,R1             ; Restore the I/O packet address to R1

        MOV     U.QACP(R5),R0     ; Get address of ACP's TCB. The offset U.ACP
                                  ; can NOT be used since file system ACP uses
                                  ; that location.  The UCB is used for TCB
                                  ; because it is easy for the ACP to access
                                  ; (as opposed to some location within the
                                  ; driver).

        BNE     10$               ; If NE the ACP has been started.

                                  ; The next few lines of code may be used as
                                  ; an alternate method of starting the ACP.
                                  ; They allow the ACP to be started
                                  ; automatically if it is installed.
                                  ; They can't be used in this application
                                  ; since ACP needs some initialization info

        .IF DF  AUTOST            ; If defined, auto start ACP

        MOV     #ACPTNM,R3        ; Get address of ACP task name
        CALL    $SRSTD            ; See if our ACP is installed
        BCS     20$               ; If CS no
        BIT     #T3.ACP,T.ST3(R0) ; Built as an ACP?
        BEQ     20$               ; If EQ no
        MOV     R0,U.QACP(R5)     ; Save address of ACP

        .ENDC

10$:    INC     U.QTRN(R5)        ; Increment count of transactions in ACP queue
        JMP     $EXRQP            ; Queue to the ACP by priority and activate
                                  ; it.  This request will unstop the ACP if
                                  ; is stopped or run it if its not active.

20$:                              ; Reference label
        MOV     #IE.OFL&377,R0    ; I/O status of device not ready
        BR      IOFIN             ; Finish I/O request

ERRIFC: MOV     #IE.IFC&377,R0    ; I/O status of illegal function code
                                  ; Finish I/O request

IOFIN:  CLR     R1                ; Second I/O status word
        JMP     $IOFIN            ; Finish I/O request

        .DSABL  LSB
```

```
;
; **-QDPWF-Powerfail, Mark offline units offline
;

QDPWF:  BIT     #UQ.ONL,U.QCTL(R5) ; Unit offline?
        BNE     10$                ; If NE no
        BISB    #US.OFL,U.ST2(R5)  ; Set offline
10$:                               ; Reference label

;
; **-QDCAN-Cancel I/O in progress, ignored
; **-QDOUT-Device timeout, does not apply
;

QDCAN:
QDOUT:
EXIT:   RETURN                     ; These functions are no-ops

        .END



        .TITLE  QDACP
        .IDENT  /01/

        .ENABL  LC

;
; This ACP is purely for purposes of example.  It is not intended to
; be a useful ACP nor is it supported in any way.  It is, however,
; functional, complete, and representative of a valid interface.
;

;
; **-QDACP-QD: driver ACP
;
;

;
; MACRO LIBRARY CALLS
;

        .MCALL  ALUN$S,DIR$,QIO$,WTSE$,WSIG$S

;
; LOCAL DATA
;

.QIO::  QIO$    ,1,1,,IOSB,,<,,,,,> ; My own QIO DPB

.IOST:: .BLKW   2                  ; I/O status to return to user
.IOPKT::.BLKW   1                  ; Address of current I/O packet
.ACTUN::.WORD   0                  ; Count of active units

WTSE:   WTSE$   1                  ; Wait for I/O completion

IOSB:   .BLKW   2                  ; I/O status block for my I/O

FID:    .BLKW   3                  ; File ID of work file
```

```
;
; **-.START-ACP starting entry point
;

.START::CALL    .INIT           ; Do one-time initialization

;
; **-.GTPKT-Get the next I/O packet from ACP queue and dispatch function
;

.GTPKT::CLR     .IOPKT          ; No I/O packet yet
        SWSTK$  30$             ;; Switch to system state to synchronize with
                                ;; Executive.  This prevents context switching
                                ;; and makes Executive routines accessable.
                                ;; On return from system state, execution will
                                ;; resume at 30$.  This call also saves all
                                ;; registers.
        MOV     $TKTCB,R0       ;; Address of my TCB (must be my TCB since
                                ;; can't execute in context of any other task
        ADD     #T.RCVL,R0      ;; Point to receive queue listhead
        CALL    $QRMVF          ;; Attempt to dequeue I/O packet from queue
        BCC     20$             ;; If CC I/O packet removed from queue
        TST     .ACTUN          ;; Is this ACP still active for any units?
        BNE     10$             ;; If NE yes
        MOV     $TKTCB,R5       ;; R5 must be our TCB address
        JMP     $DREXT          ;; No I/O requests in our queue and no active
                                ;; units, perform a task exit without any
                                ;; possibility of a race between QDCON
                                ;; inserting an I/O request in our queue
                                ;; and our task exit.
10$:    JMP     $STPCT          ;; Stop current task (us) and return to task
                                ;; state.  Once back in task state the PC will
                                ;; be at 30$, since once we are unstopped we will
                                ;; resume execution at 30$, not the next line.
20$:    MOV     R1,.IOPKT       ;; Save I/O packet address.  (Return to task
                                ;; state restores all registers.)
        RETURN                  ;; Return to task state.  Complementary to
                                ;; SWSTK$.
30$:    MOV     .IOPKT,R3       ; Get I/O packet address
        BEQ     .GTPKT          ; If EQ none found in queue, try again since
                                ; someone unstopped us.
        MOV     I.UCB(R3),R5    ; Get UCB address for request

;
; Process I/O request
;
; Do any initialization required
;

        MOV     #IS.SUC,.IOST   ; Initial status of success
        CLR     .IOST+2         ; ...
        MOV     #.QIO+Q.IOPL,R0 ; Point to parameter list are of our QIO DPB
        MOV     #6,R1           ; Number of words to clear
40$:    CLR     (R0)+           ; Clear them
        DEC     R1              ; Done?
        BNE     40$             ; If NE no
        MOV     U.QLUN(R5),.QIO+Q.IOLU ; Setup LUN in DPB
```

```
;
; Dispatch function
;
; I/O function is dispatched with the following registers
;
;       R5=UCB Address
;       R3=I/O Packet
;
; And .QIO has the correct LUN plugged into the DPB
;

        MOVB    I.FCN+1(R3),R0   ; Get I/O function code
        CMPB    #IO.RLB/400,R0   ; Read logical block?
        BNE     100$             ; If NE no
        JMP     FCRLB            ; Process read
100$:   CMPB    #IO.WLB/400,R0   ; Write logical block?
        BNE     110$             ; If NE no
        JMP     FCWLB            ; Process write
110$:   CMPB    #IO.CTL/400,R0   ; ACP control function?
        BEQ     FCCTL            ; If EQ yes

;
; Illegal function code
;

IEIFC:  MOV     #IE.IFC&377,.IOST ; I/O status of illegal function code

;
; **-.IOFIN-Finish I/O request returning status to user.
;
; INPUTS:
;       .IOST=I/O status of current request
;       .IOPKT=Address of I/O packet
;

.IOFIN: MOV     .IOST,R0         ; Get I/O status
        MOV     .IOST+2,R1       ; ...
        MOV     .IOPKT,R3        ; Get address of I/O packet
        MOV     I.UCB(R3),R5     ; Get UCB address
        SWSTK$  20$              ;; Switch to system state
        CLR     .IOPKT           ;; No I/O packet anymore
        DEC     U.QTRN(R5)       ;; Decrement count of outstanding I/O queued
                                 ;; to ACP
        BNE     10$              ;; If NE more requests in queue
        BIT     #UQ.STP,U.QCTL(R5) ;; Has a request to stop processing on
                                 ;; this unit been received?
        BEQ     10$              ;; If EQ no
        BITB    #US.MNT,U.STS(R5) ;; Unit still mounted?
        BEQ     10$              ;; If EQ yes
        TST     U.ATT(R5)        ;; Unit attached?
        BNE     10$              ;; If NE yes
        BIC     #UQ.STP!UQ.ONL,U.QCTL(R5) ; Clear our on-line bit and stop
                                 ;; request flag
        BISB    #US.OFL,U.ST2(R5) ;; Mark unit offline
        CLR     U.QACP(R5)       ;; No ACP active on unit
        INC     .IOPKT           ;; Flag to indicate unit went to off-line state
10$:    JMP     $IOFIN           ;; Finish I/O request and return to task state

20$:    ASR     .IOPKT           ; Unit go to offline state?
        BCC     30$              ; If CC no
        CALL    .CLOSE           ; Close up shop on this unit
        DEC     .ACTUN           ; Decrement count of active units
30$:    JMP     .GTPKT           ; Get next I/O request
```

```
;
; **-.DOIO-Do I/O for user to real device
;
; This routine maps to the users buffer(s) and issues an I/O request that
; will use the requesting task's buffers.  This occurs because the QIO
; processing uses the logical mapping, not the virtual mapping, to determine
; where buffers are located.  By logical mapping, I mean the physical mapping
; contained in the memory management registers.  Because we are privileged,
; no validity checking is made on the buffers, so it is possible to do I/O
; to buffers larger than the windows through which they are mapped, that is, greater
; than about 4KB.  This routine will function properly if the ACP overmaps
; the I/O page because it switches to the system stack, hence to kernel mode.
; Therefore, this routine must not be mapped by APR 7.
;
; INPUTS:
;       R0=Buffer 1 mapping for user APR 1
;       R1=Buffer 2 mapping for user APR 2
;
; OUTPUTS:
;       I/O issued and completed
;       If CC then IOSB is I/O status
;       If CS then $DSW is directive error
;

.DOIO:  SWSTK$  10$             ; Switch to system state
        MOV     UISAR0+2,2(SP)  ; Save user APR 1 value in saved R0
        MOV     UISAR0+4,4(SP)  ; And user APR 2 value in saved R1
        MOV     R0,UISAR0+2     ; Map to user's first buffer
        MOV     R1,UISAR0+4     ; Map to user's second buffer
        INCB    $CXDBL          ; Disable context switching
        RETURN                  ; Return to task state
10$:                            ; Reference label

;
; Issue the QIO directive, context switching is disabled so we don't have
; to worry about user APRs 1 and 2 being modified.  However, we can't wait
; for the I/O to complete at this point.
;

        DIR$    #.QIO           ; Issue I/O request
        ROR     -(SP)           ; Save carry state

;
; Buffers have been "validated" and relocated so we can restore original
; mapping and enable context switching.
;

        SWSTK$  20$             ; Switch to system state
        MOV     R0,UISAR0+2     ; Restore user APR 1
        MOV     R1,UISAR0+4     ; Restore user APR 2
        DECB    $CXDBL          ; Enable context switching
        RETURN                  ; Return to user state
20$:                            ; Reference label

;
; Context switching is now enabled, check directive status and if successful
; wait for completion
;

        ROL     (SP)+           ; Restore carry, was directive successful?
        BCS     30$             ; If CS no
        DIR$    #WTSE           ; Wait for I/O to complete
30$:    RETURN                  ; Return to caller
```

```
;
; **-.BLXI-Transfer data into our buffer
;
; INPUTS:
;       R0=Byte count to transfer
;       R1=Address mapping base
;       R2=APR 6 displacement
;       R3=Address of our buffer
;
; OUTPUTS:
;       Data in our buffer
;
        .ENABL  LSB

.BLXI:  SWSTK$  20$                 ; Switch to system state
        MOV     R0,R5               ; Save R5
        MOV     R1,-(SP)            ; Save R1
        MOV     R2,-(SP)            ; And R2
        MOV     R3,R0               ; Set virtual address of our buffer
        CALL    $RELOC              ; Convert to address double word
        MOV     R1,R3               ; Copy R1 and
        MOV     R2,R4               ; R2 to proper place for $BLXIO
        MOV     (SP)+,R2            ; Restore R2
        MOV     (SP)+,R1            ; and R1
        BR      10$                 ; Join common code

;
; **-.BLXO-Transfer out of our buffer into user's buffer
;
; INPUTS:
;       R0=Byte count to transfer
;       R1=Address mapping base
;       R2=APR 6 displacement
;       R3=Address of our buffer
;
; OUTPUTS:
;       Data in user buffer
;

.BLXO:  SWSTK$  20$                 ; Switch to system state
        MOV     R0,R5               ; Save R0
        MOV     R3,R0               ; Set R0 to address of our buffer for $RELOC
        MOV     R1,R3               ; Copy R1 and
        MOV     R2,R4               ; R2 into proper place for $BLXIO
        CALL    $RELOC              ; Convert to address doubleword
10$:    MOV     R5,R0               ; Restore byte count
        ADD     #120000-140000,R2 ; Convert to APR 5 displacement
        JMP     $BLXIO              ; Transfer data and return to task state
20$:    RETURN                      ; Return to caller

        .DSABL  LSB
```

```
;
; **-FCCTL-ACP control functions
;
; Two control functions are supported, start-up and shut-down.
;
; We check the request for validity and then set up various fields in the
; drivers UCB.
;

FCCTL:  MOV     I.TCB(R3),R0       ; Get TCB address of issuing task
        BIT     #T3.PRV,T.ST3(R0)  ; Task privileged?
        BEQ     IEIFC              ; If EQ no
        CMPB    #1,I.FCN(R3)       ; Startup request?
        BEQ     10$                ; If EQ yes
        CMPB    #2,I.FCN(R3)       ; Stop request?
        BEQ     30$                ; If EQ yes
        BR      IEIFC              ; Illegal function

; Process start request

10$:    TST     U.QACP(R5)         ; Already got an ACP?
        BNE     20$                ; If NE yes
        CALL    .OPEN              ; Open up shop for unit
        BCS     100$               ; If CS failed to open channel to "device"
        INC     .ACTUN             ; Increment count of units active
        MOV     $TKTCB,U.QACP(R5)  ; Set ACP TCB address in UCB
        BIS     #UQ.ONL,U.QCTL(R5) ; Set unit online
        BICB    #US.OFL,U.ST2(R5)  ; And for the operating system
        BR      100$               ; Join common code
20$:    MOV     #IE.RSU&377,.IOST  ; ACP already started for unit error
        BR      100$               ; Join common code

; Process stop request

30$:    CMP     $TKTCB,U.QACP(R5)  ; Unit online with correct ACP?
        BNE     50$                ; If NE no
        BIT     #UQ.STP,U.QCTL(R5) ; Stop requested?
        BNE     60$                ; If NE yes
        CALL    $SWSTK,100$        ;; Go to system state to prevent a race problem
        BITB    #US.MNT,U.STS(R5)  ;; Unit still mounted?
        BEQ     40$                ;; If EQ yes
        TST     U.ATT(R5)          ;; Unit attached?
        BNE     40$                ;; If NE yes
        CMP     #1,U.QTRN(R5)      ;; Is this this the only transaction queued?
        BNE     40$                ;; If NE no
        BISB    #US.OFL,U.ST2(R5)  ;; Mark unit off line to prevent further I/O
        BIS     #UQ.STP,U.QCTL(R5) ;; Request unit be stopped
        RETURN                     ;; Return to task state at statement 100$
40$:    MOV     #IE.NFW&377,.IOST  ;; Unit busy, attached, or mounted
        RETURN                     ;; Return to task state at statement 100$

50$:    MOV     #IE.OFL&377,.IOST  ; Unit offline
        BR      100$               ; Join common code

60$:    MOV     #IE.FLN&377,.IOST  ; Already being stopped; error

100$:   JMP     .IOFIN             ; Finish I/O request
```

```
;
;                            W A R N I N G
;
;       The above code must be in the first 8K of the ACP because
;       a switch to system state uses the kernel mode mapping which
;       allows only 8K for task mapping. (The I/O page is mapped in
;       thru APR 7 in system state, but may be overlaid by the task
;       in task state.)
;
;


;
; **-FCRLB-Read logical block function
;

        .ENABL  LSB

FCRLB:  CALL    .READ               ; Data in memory already?
        BCS     10$                 ; If CS no
        MOV     I.PRM+4(R3),R0      ; Get byte count
        MOV     R0,.IOST+2          ; Set return status byte count
        MOV     I.PRM(R3),R1        ; APR mapping
        MOV     I.PRM+2(R3),R2      ; APR6 displacement
        MOV     R4,R3               ; Address of our buffer
        CALL    .BLXO               ; Transfer to user buffer
        BR      30$                 ; Join common exit code

;
; **-FCWLB-Write logical block function
;

FCWLB:  CALL    .WRITE              ; Transfer data to our buffer first?
        BCS     10$                 ; If CS no
        MOV     I.PRM+4(R3),R0      ; Get byte count
        MOV     I.PRM(R3),R1        ; APR mapping
        MOV     I.PRM+2(R3),R2      ; APR6 displacement
        MOV     R4,R3               ; Address of our buffer
        CALL    .BLXI               ; Transfer to our buffer
        MOV     .IOPKT,R3           ; Restore I/O packet address

;
; Do I/O from user buffer
;

10$:    MOV     I.PRM(R3),R0        ; Get mapping value for APR 1
        MOV     I.PRM+2(R3),R1      ; Get displacement biased for APR6
        SUB     #140000-20000,R1 ; Adjust to an APR1 bias
        MOV     R1,.QIO+Q.IOPL      ; Insert virtual address of buffer when mapped
                                    ; via APR 1
        CALL    .DOIO               ; Issue I/O request
        BCC     20$                 ; If CC successful
        MOV     #IE.ABO&377,.IOST ; Return error to user
        BR      30$                 ; Join common code
20$:    MOV     IOSB,.IOST          ; Return status to user
        MOV     IOSB+2,.IOST+2      ; ...
30$:    JMP     .IOFIN              ; Finish I/O request and dispatch next request

        .DSABL  LSB
```

```
;
; **-.INIT-One time initializations on startup
;

.INIT:                          ; None
        RETURN                  ;


;
; **-.OPEN-Open up I/O path for unit
;
; Inputs:
;       R5=UCB address
;       R3=I/O packet address
;

.OPEN:  ALUN$S  U.QLUN(R5),#"SY,#0 ; Assign LUN to work file device
        BCS     20$             ; If CS error
        MOV     #IO.CRE,.QIO+Q.IOFN ; Setup for create file
        MOV     #FID,.QIO+Q.IOPL ; Insert address to receive file ID
        MOV     #100000,.QIO+Q.IOPL+4 ; Enable extend
        MOV     I.PRM(R3),.QIO+Q.IOPL+6 ; Allocate file of size requested
        CALL    XIO             ; Create and extend file
        MOVB    IOSB,.IOST      ; Copy I/O status
        BMI     10$             ; If MI error
        CLR     .QIO+Q.IOPL+4   ; Reset parameter
        CLR     .QIO+Q.IOPL+6   ; Ditto
        MOV     #IO.ACW,.QIO+Q.IOFN ; Set up to access the file
        MOV     #100000,.QIO+Q.IOPL+10 ; Enable access
        CALL    XIO             ; Access file
        MOVB    IOSB,.IOST      ; Copy I/O status
        BMI     10$             ; If MI error
        CLR     .QIO+Q.IOPL+10  ; Reset parameter
        MOV     #IO.DEL,.QIO+Q.IOFN ; Set up to mark file for delete
        CALL    XIO             ; Mark file for delete
        MOV     I.PRM(R3),U.CW3(R5) ; Set up device size
        RETURN                  ; Successful exit with carry clear
10$:    SEC                     ; Error exit with carry set
        RETURN                  ;

20$:    CRASH                   ; Internal error

;
; **-.CLOSE-Close channel to device
;

.CLOSE: MOV     #6,R0           ; Number of parameters to clear
        MOV     #.QIO+Q.IOPL,R1 ; Address of parameter list
10$:    CLR     (R1)+           ; Reset parameters
        DEC     R0              ; Done?
        BNE     10$             ; If NE no
        MOV     #IO.DAC,.QIO+Q.IOFN ; Set to deaccess file
        JMP     XIO             ; Deaccess file

;
; **-.READ-Determine method of performing read
;
; Inputs:
;       R5=UCB Address
;       R3=I/O Packet
```

```
;
; Outputs:
;       If CS then do I/O directly into user buffer
;               .QIO DPB setup with I/O function code
;       If CC then do I/O to our buffer, then copy to user buffer
;               R4=Address of buffer
;

.READ:  MOV     #IO.RVB,.QIO+Q.IOFN ; Set up to read
        MOV     I.PRM+4(R3),.QIO+Q.IOPL+2 ; Insert byte count into DPB
        MOV     I.PRM+10(R3),.QIO+Q.IOPL+6 ; And block number to start transfer
        MOV     I.PRM+12(R3),.QIO+Q.IOPL+10 ;
        ADD     #1,.QIO+Q.IOPL+10 ; Convert from "logical" to "virtual"
        ADC     .QIO+Q.IOPL+6    ;
        CMP     #1000,I.PRM+4(R3) ; Do I/O to our buffer first?
        BNE     10$              ; If NE no
        MOV     #.BUF,R4         ; Set address of buffer
        MOV     R4,.QIO+Q.IOPL   ; Set up buffer address
        CALL    XIO              ; Read data into our buffer
        TSTB    IOSB             ; On error go directly to user buffer; error?
        BMI     10$              ; If MI yes
        CLC                      ; Copy to user buffer
        RETURN                   ;
10$:    SEC                      ; Do I/O directly to user buffer
        RETURN                   ;

;
; **-.WRITE-Determine method of performing write
;
; Inputs:
;       R5=UCB Address
;       R3=I/O Packet
;
; Outputs:
;       If CS then do I/O directly from user buffer
;               .QIO DPB setup with I/O function code
;       If CC then copy data to our buffer, then do I/O from user buffer
;               R4=Address of buffer
;

.WRITE: MOV     #IO.WVB,.QIO+Q.IOFN ; Set up to write
        MOV     I.PRM+4(R3),.QIO+Q.IOPL+2 ; Insert byte count into DPB
        MOV     I.PRM+10(R3),.QIO+Q.IOPL+6 ; And block number on device
        MOV     I.PRM+12(R3),.QIO+Q.IOPL+10 ;
        ADD     #1,.QIO+Q.IOPL+10 ; Convert from "logical" to "virtual"
        ADC     .QIO+Q.IOPL+6    ;
        CMP     #1000,I.PRM+4(R3) ; Copy to our buffer first?
        BNE     10$              ; If NE no
        MOV     #.BUF,R4         ; Address of buffer for data
        CLC                      ; Copy from user buffer
        RETURN                   ;
10$:    SEC                      ; Do I/O directly from user buffer
        RETURN                   ;
```

```
;
; **-XIO-Execute QIO request
;

XIO:    DIR$    #.QIO           ; Issue I/O request
        BCC     10$             ; If CC successfully issued
        CMP     #IE.UPN,$DSW    ; No dynamic storage available?
        BNE     20$             ; If NE no
        WSIG$S                  ; Hope
        BR      XIO             ; ...hope
10$:    DIR$    #WTSE           ; Wait for I/O to complete
        BCS     20$             ; If CS error
        RETURN                  ;

20$:    CRASH                   ; Internal error

;
; I/O buffer
;

.BUF:   .BLKB   1000            ; One block long

        .END    .START




        .TITLE  QDCON
        .IDENT  /01/

        .ENABL  LC

;
; This control task is purely for purposes of example.  It is not intended to
; be a useful task nor is it supported in any way.  It is, however,
; functional, complete, and representative of a valid interface.
;


;
; **-QDCON-QD: driver and ACP control task
;
;


;
; MACRO LIBRARY CALLS
;

        .MCALL  ALUN$,GLUN$,DIR$,GMCR$,WTSE$S,QIOW$S,EXST$S
        .MCALL  ISTAT$,STATE$,TRAN$

;
; DEFINE PARSER STATE TABLE
;
; The following commands are supported:
;
;       >QDC START QDn:/SIZE:n
; where
;       START is the subcommand to startup the ACP specifying the "disk size"
;
;       >QDC STOP QDn:
; where
;       STOP is the subcommand to stop the ACP at the earliest opportunity
;
```

```
        ISTAT$  QDCSTB,QDCKTB

; Skip command name

        STATE$  INITL
        TRAN$   $STRNG

; Determine subcommand

        STATE$
        TRAN$   "START",START,,$START,$DISPT
        TRAN$   "STOP",STOP,,$STOP,$DISPT

; Process START command, get device name

        STATE$  START
        TRAN$   !DEVICE

        STATE$  OPTION
        TRAN$   '/,SWITCH
        TRAN$   $EOS,$EXIT

        STATE$  SWITCH
        TRAN$   "SIZE",SIZE

        STATE$  SIZE
        TRAN$   '=

        STATE$
        TRAN$   $NUMBR,OPTION,SETSIZ

; Process STOP command

        STATE$  STOP
        TRAN$   !DEVICE

        STATE$
        TRAN$   $EOS,$EXIT

; Process device name

        STATE$  DEVICE
        TRAN$   $ALPHA,,SETDV1

        STATE$
        TRAN$   $ALPHA,,SETDV2

        STATE$
        TRAN$   $NUMBR,DEV1,SETUNT
        TRAN$   $LAMDA

        STATE$  DEV1
        TRAN$   ':,$EXIT

; Terminate state table

        STATE$
```

```
;
; PARSER ACTION ROUTINES
;

; Set first character of device name

SETDV1: MOVB    .PCHAR,$DEV       ; Save first character of device name
        RETURN                    ;

; Set second character of device name

SETDV2: MOVB    .PCHAR,$DEV+1     ; Save second character
        RETURN                    ;

; Set device unit number

SETUNT: MOV     .PNUMB,$UNIT      ; Save converted unit number
        RETURN                    ;

; Set device size

SETSIZ: MOV     .PNUMB,$SIZE      ; Save converted size
        RETURN                    ;

;
; LOCAL DATA
;

ALUN:   ALUN$   1,,               ; Assign LUN to QDn:
$DEV=   ALUN+A.LUNA               ; Address of device name
$UNIT=  ALUN+A.LUNU               ; Address of device unit number

GLUN:   GLUN$   1,GLUBUF          ; Get LUN information for QD: device

GLUBUF: .BLKW   6                 ; LUN information buffer
$PDEV=  GLUBUF+G.LUNA             ; Actual device name
$PUNIT= GLUBUF+G.LUNU             ; Actual device unit
$CHAR=  GLUBUF+G.LUCW             ; Device characteristics word

GMCR:   GMCR$                     ; Get MCR command line

$SIZE:  .WORD   0                 ; Device size to create

$DISPT: .WORD   0                 ; Address of service routine

ACPNAM: .RAD50  /QDACP /          ; Name of ACP

PRMLST: .BLKW   8.                ; Parameter list for I/O packet

$IOST:  .BLKW   2                 ; I/O status

;
; Error messages
;

.MACRO  ERM     ERN,STS,TEXT
                .PSECT  $$ERMG
        $$$1=.
                .ASCII  <15>"TEXT"
        $$$2=.
                .PSECT
        ERN:    .WORD   STS
                .WORD   $$$1,$$$2-$$$1
.ENDM
```

```
.MACRO  ERRX    ERN
        MOV     #ERN,R0
        JMP     $ERRXT
.ENDM

.MACRO  FTLX    ERN
        MOV     #ERN,R0
        JMP     $SUCXT
.ENDM

.MACRO  SUCX    ERN
        MOV     #ERN,R0
        JMP     $SUCXT
.ENDM

ERM     ERRCML,14,<%QDC-F-GETCOMFAIL, Failed to get command line>
ERM     ERRSYN,24,<%QDC-F-SYNERR, Syntax error in command>
ERM     ERRNQD,34,<%QDC-F-NOQDDEV, Failed to assign LUN to QD:>
ERM     ERRBDV,44,<%QDC-F-BADDEVICE, Invalid device specified>
ERM     ERRNOD,54,<%QDC-F-NOPOOL, No dynamic memory for I/O request>
ERM     ERRNAC,64,<%QDC-F-NOACP,QDACP not installed in system>
ERM     ERRREQ,74,<%QDC-F-REQFAIL, Failed to request QDACP>
ERM     ERRUSE,104,<%QDC-F-DEVINUSE, Specified unit already in use>
ERM     ERRINT,114,<%QDC-F-INTERNAL, Internal error>
ERM     ERRFLN,124,<%QDC-F-OFFLINREQ, Unit already requested to offline>
ERM     ERRNOL,134,<%QDC-F-NOTONLINE, Unit not online>
ERM     ERRNDS,144,<%QDC-F-NODISSPAC, Failed to allocate disk space>
ERM     ERRBSY,154,<%QDC-F-DEVICEBUSY, Device busy, mounted, or attached>
ERM     ERRFTL,0,<-QDC-F-ONLFAIL, Failed to bring unit online>
ERM     SUCCOM,161,<%QDC-S-ONLINE, Specified unit brought online>
ERM     REQOFF,171,<%QDC-S-REQOFFLINE, Unit requested to offline>

;
; **-.QDCON-QD device control program
;

$QDCON: CLR     $DISPT          ; Reset service routine dispatch address
        CLR     $UNIT           ; Clean out unit number
        DIR$    #GMCR           ; Get the command line
        BCC     10$             ; If CC successful
        FTLX    ERRCML

10$:    CLR     R1              ; Suppress blanks
        MOV     #QDCKTB,R2      ; Get keyword table address
        MOV     $DSW,R3         ; Get length of command line
        MOV     #GMCR+G.MCRB,R4 ; Get address of command line
        MOV     #INITL,R5       ; Get address of initial parser state
        CALL    .TPARS          ; Parse command line
        BCC     20$             ; If CC good command line
        FTLX    ERRSYN

20$:    DIR$    #ALUN           ; Assign LUN to QD:
        BCC     30$             ; If CC good device
        FTLX    ERRNQD

30$:    DIR$    #GLUN           ; Get device information

        JMP     @$DISPT         ; Dispatch to START or STOP service
```

```
;
; **-$START-Start up ACP and specify size
;

$START: CMP     #"QD,$PDEV      ; Really QD:?
        BEQ     10$             ; If EQ yes
        FTLX    ERRBDV

10$:    MOV     $SIZE,PRMLST    ; Address of parameter
        MOV     #PRMLST,R4      ; Get address of parameter list
        MOV     #ACPNAM,R3      ; Get address of ACP task name
        MOV     #IO.CTL!1,R2    ; Set function code of ACP control
                                ;  and subfunction of START (1)
        CALL    .QUEIO          ; Queue an I/O request to the ACP
        BCC     100$            ; If CC successfully queued
        CMP     #IE.UPN,$DSW    ; No pool?
        BNE     20$             ; If NE no
        ERRX    ERRNOD
20$:    CMP     #IE.INS,$DSW    ; ACP not installed?
        BEQ     30$             ; If EQ yes
        CMP     #IE.PRI,$DSW    ; Not an ACP?
        BNE     40$             ; If NE no
30$:    ERRX    ERRNAC
40$:    ERRX    ERRREQ          ; Catchall error message

100$:   CMPB    #IS.SUC,$IOST   ; Success?
        BNE     110$            ; If NE no
        SUCX    SUCCOM          ; Successful completion

110$:   CMPB    #IE.RSU,$IOST   ; Already got an ACP or already started?
        BNE     120$            ; If NE no
        ERRX    ERRUSE
120$:   CMPB    #IE.DFU,$IOST   ; Device full?
        BNE     130$            ; If NE no
        ERRX    ERRNDS
130$:   FTLX    ERRINT          ; Catchall error message

;
; **-$STOP-Stop ACP operation
;

$STOP:  CMP     #"QD,$PDEV      ; Really QD:?
        BEQ     10$             ; If EQ yes
        FTLX    ERRBDV

10$:    MOV     #PRMLST,R4      ; Get address of parameter list
        MOV     #ACPNAM,R3      ; Get address of ACP task name
        MOV     #IO.CTL!2,R2    ; Set function code for ACP control
                                ;  and subfunction of STOP (2)
        CALL    .QUEIO          ; Queue an I/O request to the ACP
        BCC     100$            ; If CC successfully queued
        CMP     #IE.UPN,$DSW    ; No pool?
        BNE     140$            ; If NE no
        FTLX    ERRNOD

100$:   CMPB    #IS.SUC,$IOST   ; Success?
        BNE     110$            ; If NE no
        SUCX    REQOFF          ; Successful completion

110$:   CMPB    #IE.FLN,$IOST   ; Already being off-lined?
        BNE     120$            ; If NE no
        FTLX    ERRFLN
```

```
120$:   CMPB    #IE.NFW,$IOST     ; Unit busy?
        BNE     130$              ; If NE no
        FTLX    ERRBSY
130$:   CMPB    #IE.OFL,$IOST     ; Device not on line?
        BEQ     140$              ; If EQ yes
        FTLX    ERRINT            ; Catchall error message
140$:   FTLX    ERRNOL            ; Not on line


;
; **-.QUEIO-Create and queue an I/O packet directly to an ACP
;
; This routine builds an I/O packet and queues it directly to an
; ACP bypassing the Executive QIO directive processing.  The primary
; reason for doing this is the fact that under some circumstances
; the ACP cannot be reached by going through the driver, such as when the
; ACP has not been started.  The parameter list is copied into the I/O packet
; without modification.  Consequently, a buffer address cannot be passed
; as a parameter; it must first be relocated and the address double word
; placed in the parameter list; this routine is not designed to do that.
;
; INPUTS:
;       R4=Parameter list address
;       R3=Address of ACP task name
;       R2=I/O function code
;       LUN 1 assigned to device associated with ACP
;
; OUTPUTS:
;       If CC ACP requested and I/O complete
;               $IOST is I/O status block containing return from ACP
;       If CS ACP not requested
;               $DSW is error status
;       IE.UPN - No dynamic memory for I/O packet
;       IE.INS - ACP not installed
;       IE.PRI - Task not an ACP
;
; If entry at .QUEIO, all registers preserved
;

        .ENABL  LSB

.QUEIO: SWSTK$  40$               ;; Switch to system state
        CLR     $IOST             ;; Clear I/O status block
        CLR     $IOST+2           ;; Indicating I/O pending
        MOV     $HEADR,R5         ;; Get address of our task header
        MOV     H.LUN(R5),R5      ;; Get device UCB address
        MOV     #IE.INS,$DSW      ;; Assume task not found
        CALL    $SRSTD            ;; Search for task
        BCS     30$               ;; If CS failure
        MOV     #IE.PRI,$DSW      ;; Assume task not an ACP
        BIT     #T3.ACP,T.ST3(R0) ;; Task an ACP?
        BEQ     30$               ;; If EQ no
        MOV     R0,-(SP)          ;; Save TCB address
        MOV     R2,-(SP)          ;; Save I/O function code
        MOV     #IE.UPN,$DSW      ;; Assume buffer allocation failure
        MOV     #I.LGTH,R1        ;; Length of I/O packet
        CALL    $ALOCB            ;; Allocate buffer from pool
        BCS     30$               ;; If CS failure
        MOV     (SP)+,R3          ;; Restore R3
        MOV     R0,-(SP)          ;; Save address of I/O packet
        ASR     R1                ;; Convert size in bytes to words
10$:    CLR     (R0)+             ;; Zero I/O packet
        DEC     R1                ;; Done?
        BNE     10$               ;; If NE no
        MOV     #$IOST,R0         ;; Get I/O status block address
```

```
        CALL    $RELOC            ;; Relocate it
        MOV     (SP)+,R0          ;; Restore packet address
        MOV     #$IOST,I.IOSB(R0) ;; Insert virtual address of status block
        MOV     R1,I.IOSB+2(R0)   ;; Insert relocation bias and
        MOV     R2,I.IOSB+4(R0)   ;; Offset of I/O status block
        MOV     $TKTCB,I.TCB(R0)  ;; Insert our TCB
        MOVB    #1,I.EFN(R0)      ;; Insert event flag number
        MOVB    #251.,I.PRI(R0)   ;; Insert priority
        MOV     R5,I.UCB(R0)      ;; Insert device UCB
        MOV     R3,I.FCN(R0)      ;; Insert function code
        MOV     R0,R1             ;; Copy address of packet
        ADD     #I.PRM,R0         ;; Point to parameter area
        MOV     #8.,R2            ;; Set parameter count
20$:    MOV     (R4)+,(R0)+       ;; Copy parameter list into packet
        DEC     R2                ;; Done?
        BNE     20$               ;; If NE no
        MOV     (SP)+,R0          ;; Get ACP TCB address
        INC     U.QTRN(R5)        ;; Bump count of transactions queued to unit
        CALL    $EXRQP            ;; Queue I/O packet to ACP by priority and
                                  ;; ensure ACP is active
        MOV     $TKTCB,R0         ;; Get our TCB address
        INCB    T.IOC(R0)         ;; Bump our I/O count
        CLR     T.EFLG(R0)        ;; Clear event flag 1
        MOV     #IS.SUC,$DSW      ;; Indicate success
30$:    RETURN                    ;; Return to task state
40$:    TST     $DSW              ; QIO successful?
        SEC                       ; Assume error
        BLE     50$               ; If LE no
        WTSE$S  #1                ; Wait for I/O to finish
50$:    RETURN                    ; Return to caller

        .DSABL  LSB

;
; **-$ERRXT-Error exit
; **-$SUCXT-Success exit
;
; Inputs:
;       R0=Error table entry
;
; Outputs:
;       Message and task exit
;

        .ENABL  LSB

$ERRXT: MOV     (R0)+,R5          ; Get exit status
        MOV     (R0)+,R1          ; Get address of error text
        MOV     (R0)+,R2          ; Get size of text
        QIOW$S  #IO.WVB,#5,#5,,,,<R1,R2, 40> ; Write message
        BCC     10$
        IOT
10$:    MOV     #ERRFTL+2,R0      ; Get address of fatal error message
        BR      20$               ; Join common code
$SUCXT: MOV     (R0)+,R5          ; Get exit status
20$:    MOV     (R0)+,R1          ; Get address of final message
        MOV     (R0)+,R2          ; Get size of message
        QIOW$S  #IO.WVB,#5,#5,,,,<R1,R2, 40> ; Write message
        BCC     30$
        IOT
30$:    EXST$S  R5                ; Exit with status

        .DSABL  LSB

        .END    $QDCON
```

INDEX

INDEX

INDEX

RSX-11M
Guide to Writing
an I/O Driver
Order No. AA-2600E-TC

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) ______

Name ______ Date ______

Organization ______

Street ______

City ______ State ______ Zip Code ______
or Country

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

BSSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03061

Do Not Tear - Fold Here

Cut Along Dotted Line

Printed in U.S.A.