RSX-11M

Guide to Writing an I/O Driver

Order No. AA-2600D-TC
and
Update Notice No. 1 (AD-2600D-T1)

RSX-11 M Version 3.2

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

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

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

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

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

Copyright © 1975, 1976, 1977, 1979 by Digital Equipment Corporation

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

The following are trademarks of Digital Equipment Corporation:

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

CONTENTS

CONTENTS

CONTENTS

FIGURES

CONTENTS

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, a line 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 computer, 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.

ASSOCIATED DOCUMENTS

Familiarity with the system implies an in-depth exposure to the following RSX-11M manuals:

1. System Generation Manual
2. I/O Drivers Reference Manual
3. Executive Reference Manual
4. Utilities Procedures Manual
5. I/O Operations Reference Manual

As adjuncts to this manual, you are advised to study existing I/O drivers. The RF-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 under UIC [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 Documentation Directory. The

Documentation 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.

STRUCTURE OF THIS DOCUMENT

This document proceeds from chapter to chapter toward increasing levels of implementation detail.

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/O 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 understanding 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 PDP-11/70 NPR device drivers.

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

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, built in at SYSGEN. 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 SYSGEN dialog, you can specify that you want this facility.

Making drivers loadable can result in a smaller Executive, and thus more space for user tasks. Any driver that is not needed for a period of time can be unloaded from the system. For example, suppose your system has both a paper-tape reader and a card reader. Suppose that only one of them is connected to the system at any one time. You could load the driver for the online 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. A resident driver can only be incorporated into a system at SYSGEN; the Executive must be rebuilt and the system bootstrapped each time it is reincorporated during debugging. A loadable driver can be incorporated 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 timeout.

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 RSX-11D 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 RSX-11M's data structures. These structures 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 the centralization of functions, combined with table-driven design, enables RSX-11M to meet its intended memory and performance goals.

2.2 STRUCTURE

This section:

1. Places an I/O driver in the context of the overall RSX-11M I/O system;
2. Establishes the responsibilities of an I/O driver, and
3. Functionally describes 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 the hierarchy can be entered 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 are costly in their consumption of equipment resources. Figure 2-1 shows the loose 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.

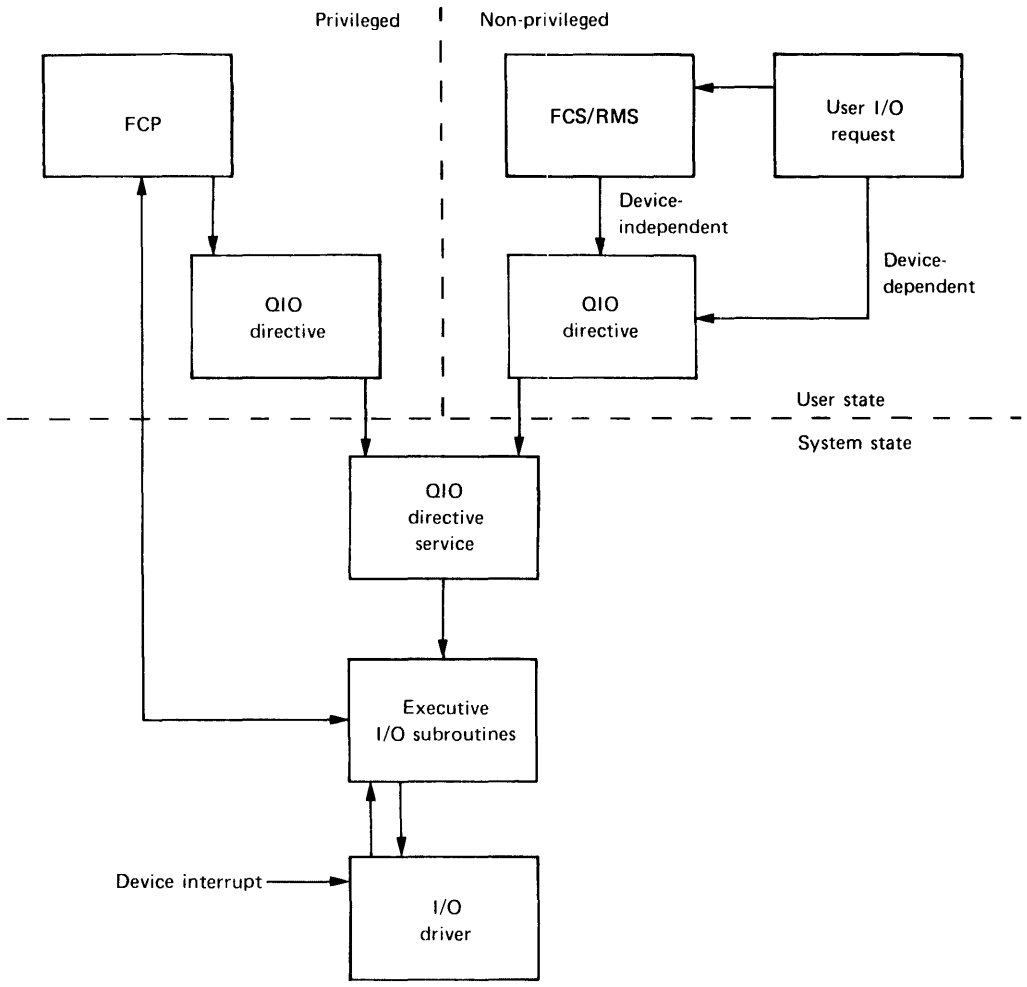


Figure 2-1 I/O Control Flow

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 may issue a QIO directive. The QIO directive 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 processing of I/O requests by the Executive I/O system is accomplished by means of:

1. File Control Primitives (FCP), and
2. A collection of Executive components consisting of:
   a. QIO directive processing;
   b. I/O-related subroutines, and
   c. The I/O drivers.

FCP is responsible for maintaining the structure and integrity of data stored on file-structured volumes. It maps virtual block numbers to logical block numbers, extends files, and makes volume-protection checks. The driver converts a logical block number into a physical address on a file-structured device. No direct connection exists between FCP and a driver.

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

The I/O services provided by the Executive consist of QIO directive processing, and a collection of subroutines used by drivers to obtain I/O requests, facilitate interrupt handling, and return status upon completion of an I/O request (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 five entry points:

1. Device interrupt*
2. I/O initiator
3. Device timeout
4. Cancel I/O
5. 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.

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.

---

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

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 wakeup signal for the driver.

2.2.2.3 **Device Timeout** - When a driver initiates an I/O operation, it can establish a timeout 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:

1. When power is restored after a failure
2. When the system is bootstrapped
3. 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 (refer to 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, because recovery will eventually be effected by means of the timeout 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 pre-operational initialization.

Load - The MCR command LOA[D] calls a loadable driver at its power-failure entry point if the device is online and UC.PWF (refer to Section 4.1.4.1) is set.

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

1. Servicing device interrupts
2. Initiating I/O operations
3. Cancelling in-progress I/O operations
4. Performing device-related functions upon the occurrence of timeout or power failure

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 FCP.

2.3 DATA STRUCTURES

An I/O driver interacts with seven data structures:

1. Device Control Blocks (DCBs)
2. Unit Control Blocks (UCBs)
3. Status Control Blocks (SCBs)
4. The I/O Packet
5. The I/O Queue
6. The Fork List
7. 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 to convey the complete interaction of an I/O driver with the Executive, we will also describe their role in the system. 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.* For example, the redirect pointer, which reflects the results of an MCR Redirect command, is updated dynamically. As with the DCB, most of the UCB is established in the assembly source; however, its contents are used and modified by both the Executive and the driver, though modification of a given datum is usually done by either the Executive or the driver, but not both.

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 RK11 Controller). Line multiplexers such as the DH11 and DJ11 are considered to have one controller for each line because all lines may 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 represents the interrelationship among the DCB, UCB, and SCB without entering into the detailed contents of the control blocks, leaving such a discussion to be pursued 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.

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.

---

* 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.

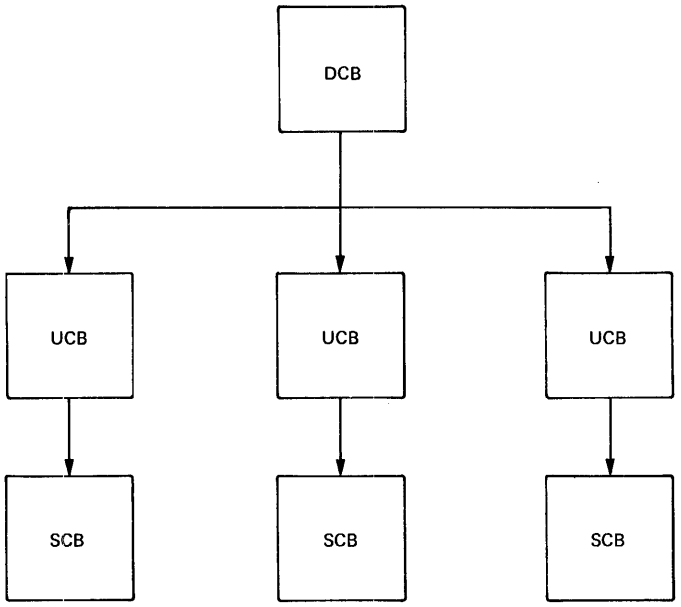


Figure 2-2 DH11 Terminal I/O Data Structure

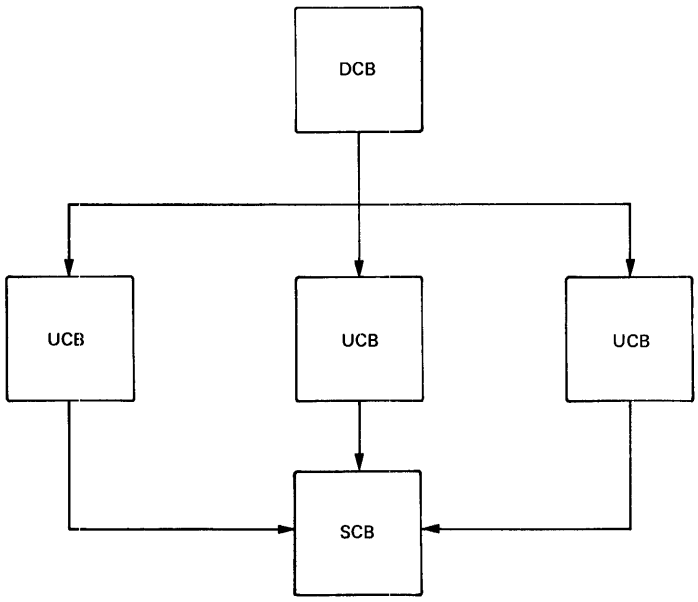


Figure 2-3 RK11 Disk I/O Data Structure

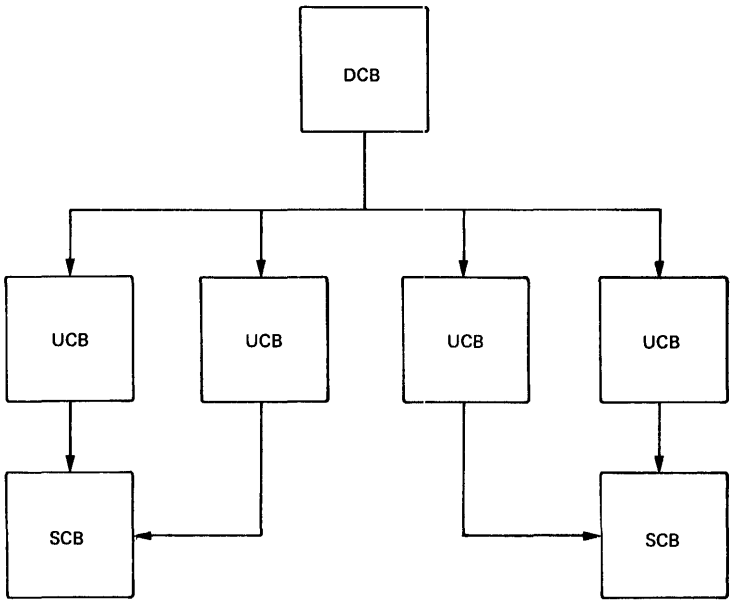


Figure 2-4 I/O Data Structure for Two RK11 Disk Controllers

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.*

---

* 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.

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 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 elapsed.

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 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 (FIFO) whenever the system stack is empty.

A process can also call $FORK to access a shared data base--a system table, for example. Such access must be strictly controlled 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 4 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 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 pre-initiation 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 thru 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--described in 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:

1. Interrupt Save ($INTSV)
2. Get Packet ($GTPKT)
3. Create Fork Process ($FORK)
4. I/O Done ($IODON or $IOALT)

2.4.2.1 **Interrupt Save ($INTSV)*** - 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.** 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.

---

* A loadable driver on a mapped system may not call $INTSV directly. See Section 4.3.

** 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.

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)** - Synchronization of access to shared data bases is accomplished 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.

An interrupt routine may not 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). Further, 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, resets the SCB and device unit status to "idle," and releases 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. Failure to observe the internal conventions and protocol can result in poor service and reductions 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 IAS/RSX-11 MACRO-11 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* 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 un-nesting 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 linearizes access to shared system data bases. 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 (PS) set with a priority of 7. It is a system software convention to use the low-order 4 bits of the PS word to encode the controller number for multicontroller drivers. When a device interrupt occurs, the hardware pushes the current PS and PC onto the current stack and loads the new PC and PS (set at PR7 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:

---

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

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, or

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 Interrupt Save 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 Interrupt Save routine. The driver sets this priority level to that of the interrupting source.

The Interrupt Save 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 Interrupt Save 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 Interrupt Save routine should not exceed 500 microseconds*.

NOTE

In actual practice, every driver in the system calls the Interrupt Save routine on every interrupt. This practice is due to two factors:

1. It is difficult to service an interrupt without using one or two registers.

---

* 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. Most interrupt code can safely be executed at the priority of the interrupting source. Execution at that priority is more desirable in terms of system response than continuing to execute at the highest priority.

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.

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 500us.

3. All modifications to system data bases must be done by a fork process.

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 described below:

1. [Task issues QIO directive]

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

1a. [First-level validity checks]

   The QIO directive processor validates the LUN and UCB pointer.

1b. [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.

1c. [Additional validity checks]

   The EFN is validated, as well as the address of the I/O Status Block (IOSB). The event flag is reset and the I/O status block is cleared.

2. [Obtain storage for and create 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. [Validate the function requested]

   The function is one of four possible types:

   Control

   No-op

   ACP

   Transfer

   Control functions, with the exception of Attach/Detach, are queued to the driver. If the bit UC.ATT is set, Attach/Detach will also be queued to the driver.

   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).

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

4. [Driver processing]

4a. [Request work]

To obtain work, the driver calls Get Packet ($GTPKT). $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.

4b. [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.* (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).

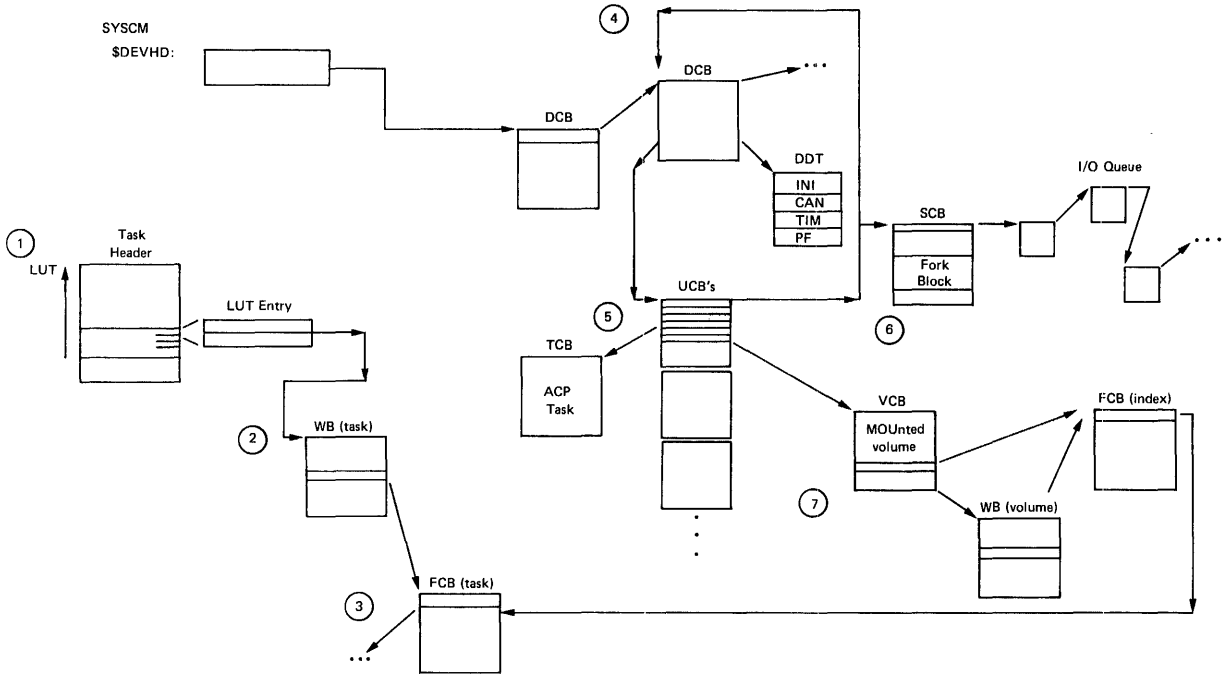


Figure 2-5 I/O Data Structure

* In mapped systems, a copy of the Task Header (located in the task's partition) is made in the Executive's dynamic storage region. This copy is then used by the Executive. To access the current information in this copy, a task must be privileged and have mapping to the Executive.

2. A Window Block (WB) exists for each active access to files on a mounted volume. Its function is to speed up the process of retrieving data items in the file; entries in a WB consist mainly of pointers to contiguous areas on the device on which the file resides. The driver is not concerned with the WB.

3. Each uniquely opened file on a mounted volume has an associated File Control Block (FCB). The file system uses the FCB to control access to the file.

4. $DEVHD is a word located in system common (SYSCM) and 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. It contains pointers to the File Control Block (FCB) and Window Block (WB), which control access to the volume's index file. (The index file is a file of file headers.) The WB for the index file serves the same function as the WB for a user file. (See the IAS/RSX-11 I/O Operations Reference Manual 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. While we have 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 CONNECT TO INTERRUPT VECTOR directive. Refer to the description of the CINT$ directive in the RSX-11M Executive Reference Manual. For examples of the use of CINT$, examine the task-level support routines for K-series laboratory peripheral modules.

3.1 OVERVIEW OF USER-WRITTEN DRIVERS

Incorporating a user-written driver is accomplished by means of the standard system generation process. Phase 1 of system generation includes queries that condition Phase 2 for the inclusion of user-written drivers. Specifically, the query

ARE YOU PLANNING TO INCLUDE A USER WRITTEN DRIVER? [Y/N]:

if answered affirmatively, results in at least one additional query. This query is:

WHAT IS THE ADDRESS OF THE HIGHEST DEVICE INTERRUPT VECTOR? [O]:

Phase 1 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).

The following two additional queries may also appear (refer to Section 5.3):

IS THE EXECUTIVE ROUTINE $GTWRD REQUIRED? [Y/N]:

IS THE EXECUTIVE ROUTINE $PTWRD REQUIRED? [Y/N]:

NOTE

These additional queries are suppressed when certain standard drivers, which require the $GTWRD and $PTWRD routines, are included during SYSGEN.

During the same phase of system generation, there is an additional decision you must make on behalf of the driver you are planning to incorporate into the system. Your driver, similar to most (but not all) DIGITAL supplied drivers, can be resident or loadable. Loadable drivers require extra Executive features to support them (for example, the MCR/VMR LOAD and UNLOAD commands). You can choose support for loadable drivers by answering in the affirmative the following Phase 1 system generation question:

DO YOU WANT LOADABLE DRIVER SUPPORT? [Y/N]:

3.1.1 Overview of User-Written Driver Code

When you decide to add a driver to your system, you share responsibility for the integrity of the Executive. Errors in your driver code can cause a system failure and bring to a halt all user service.

To create the source code to drive a device, you must perform these steps:

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:
   a. Initiator
   b. Cancel I/O
   c. Timeout
   d. Powerfail
   e. Interrupt
6. Create the driver source code. This code will contain the following global symbols (where xx is the 2-character 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).

Loadable drivers have one 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 (refer to Section 4.3) uses this symbol (and others in RSXMC.MAC) to determine if the call to $INTSV should be omitted from the driver.

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

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

When adding a resident driver to the system, you should assemble the driver with padding space for possible expansion during the debugging process. This padding space is necessary because the system may crash upon exiting VMR if the new Executive is larger than the old one (see the note in Section 3.4.1.5). For a loadable driver, however, you do not need to include padding space in the assembly source.

3.1.2 Overview of User-Written Driver Data Bases

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

1. The DCB
2. The UCB(s)
3. The SCB(s)
4. 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. The following three tables list these required fields.

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 0 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 4 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 0.

Table 3-2
Required UCB Fields

Offset	Description
U.LUIC	Log-on UIC. This field is located at a negative offset from the start of the UCB. It is present in terminal UCBs on multiuser systems only.
U.OWN	Owning terminal UCB address. This field is located at a negative offset from the start of the UCB. It is present on multiuser systems only.
U.DCB	Backpointer to the associated DCB.
U.RED	Redirect pointer--initially contains the address of this UCB.

(continued on next page)

Table 3-2 (Cont.)
Required UCB Fields

Offset	Description
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 (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 (initially 0).

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-11/70 in extended-addressing (22-bit) mode.

3.2 USER-WRITTEN RESIDENT DRIVERS

The general procedure for incorporating a user-written resident driver into your system is as follows:

1. Bootstrap the source disk and run SYSGEN Phase 1.
2. Bootstrap the object disk.
3. Create the assembly source for the driver.
4. Create the assembly source for the driver's data base.
5. Run SYSGEN Phase 2.

The following subsections present details of steps 4 and 5 above.

3.2.1 Creating the Data Base for a Resident Driver

1. Use UIC [200,200] to create source code for your driver's data base on the object disk.
2. Use USRTB.MAC as the filename and file type of the assembly source file. USRTB as a filename is not actually required. It is, however, a useful convention--one that you will see reflected in the sample dialog in Section 3.2.2.
3. 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--for examples see the file [11,10] SYSTB.MAC, created by Phase 1 of SYSGEN. 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).
4. Initialize the device interrupt vector (refer to Section 4.1.5 for description of this process).
5. Use the global label $USRTB:: as the address of your first (or only) DCB. This is absolutely required.

3.2.2 Incorporating a Resident Driver

During the execution of system generation Phase 2, the query

```
*DO YOU WISH TO ADD USER WRITTEN DRIVER(S) TO THE EXEC? [Y/N]:
```

is posed. If the answer is affirmative, then subsequent dialog guides you through the process of adding the driver to the generated system. Operations performed include assembly of the driver and its data structure, inclusion of the resultant object modules into the Executive object module library, and editing of the RSX-11M task-build command file.

The following sample dialog illustrates the addition of a driver for device XX. All user responses are underlined. The notation [1,2x] indicates that the appropriate UIC is to be substituted: [1,20] for an unmapped system and [1,24] for a mapped system.

```
>* DO YOU WISH TO ADD USER WRITTEN DRIVER(S) TO THE EXEC? [Y/N]:Y
>* WAS LOADABLE DRIVER SUPPORT SELECTED DURING SYSGEN PHASE 1? [Y/N]:Y
>SET /UIC=[200,200]
>;
>; WE WILL PAUSE WHILE YOU ASSEMBLE YOUR DRIVER(S) AND USRTB
>; MODULE.  THE EXEC ASSEMBLY PREFIX FILE RSXMC.MAC IS ALREADY
>; LOCATED UNDER UIC [200,200].  ASSUMING YOUR DRIVER(S) NAME IS
>; XXDRV, WHERE XX IS THE DEVICE NAME (E.G., DK) THE FOLLOWING
>; COMMANDS WILL ASSEMBLE THE DRIVER(S) AND THE USRTB MODULE.
>;
>;       >RUN $MAC
>;       MAC>XXDRV=[1,1]EXEMC/ML,[200,200]RSXMC,XXDRV
>;       MAC>USRTB=[1,1]EXEMC/ML,[200,200]RSXMC,USRTB
>;       MAC>^Z
>;
>
AT. -- PAUSING.  TO CONTINUE TYPE "RES ...AT."
>RUN $MAC
MAC>XXDRV=[1,1]EXEMC/ML,[200,200]RSXMC,XXDRV
MAC>USRTB=[1,1]EXEMC/ML,[200,200]RSXMC,USRTB
MAC>^Z

>RES ...AT.
AT. -- CONTINUING

>;
>; NOW YOU MUST ADD YOUR DRIVER(S) AND USRTB MODULE
>; TO THE EXEC OBJECT MODULE LIBRARY.  THE FOLLOWING IS AN EXAMPLE:
>;
>;       LBR>RSX11M/RP=[200,200]XXDRV,USRTB
>;       LBR>^Z
>;
>SET /UIC=[1,2x]
>LBR
LBR>RSX11M/RP=[200,200]XXDRV,USRTB
LBR>^Z

>;
>; YOU MUST NOW ADD COMMANDS TO INCLUDE YOUR DRIVER(S) AND USRTB
>; MODULE INTO THE EXEC BY EDITING THE EXEC TASK BUILD COMMAND FILE.
>; TO ADD DRIVER(S), INSERT COMMANDS OF THE FORM:
>;
>;       RSX11M/LB:XXDRV
>;
>; INTO THE COMMAND FILE IN THE PLACE WHERE THE
>; OTHER DRIVERS ARE REFERENCED.  XXDRV REPRESENTS THE NAME OF
>; YOUR DRIVER(S).
>;
>;             NOTE: FOR THOSE DRIVERS WHICH YOU WANT TO BE LOADABLE,
>;             DO NOT INCLUDE CORRESPONDING COMMANDS TO ADD THEM TO
>;             THE EXECUTIVE.
>;
>; THEN LOCATE THE LINE IN WHICH THE MODULE SYSTB IS
>; REFERENCED AND ADD THE ENTRY FOR YOUR
>; USRTB MODULE IMMEDIATELY AFTER IT.  EG:
>;
>;       [1,2x]RSX11M/LB:LOADR:NULTK:SYSTB:USRTB:SYTAB:INITL
>;
>; FINALLY, LOCATE THE LINE:
```

```
>;
>;       GBLDEF=$USRTB:0
>;
>; AND DELETE IT.
>;
>EDI RSXBLD.CMD
[PAGE     1]
*PL TTDRV
[1,2x]RSX11M/LB:TTDRV
*I
[1,2x]RSX11M/LB:XXDRV
↵
*PL SYSTB
[1,2x]RSX11M/LB:LOADR:NULTK:SYSTB:SYTAB:INITL
*C/SYSTB/SYSTB:USRTB/
[1,2x]RSX11M/LB:LOADR:NULTK:SYSTB:USRTB:SYTAB:INITL
*PL $USRTB
GBLDEF=$USRTB:0
*D
*EX
[EXIT]
>;
>; YOUR NON-LOADABLE DRIVERS WILL AUTOMATICALLY BE LINKED
>; WITH THE EXECUTIVE YOU ARE BUILDING.
```

This completes the user-written resident driver section of Phase 2, which then continues.

3.3 USER-WRITTEN LOADABLE DRIVERS

The procedure for incorporation of a user-written loadable driver depends on the nature of the data base associated with the driver. The data base for such a driver can be either resident or loadable.

In deciding whether the data base for your loadable driver will be resident or loadable, you should consider the following limitations on loadable data bases:

1. A loadable data base is only loaded once; thereafter it is resident until the system is bootstrapped again. The UNL[OAD] command does not remove a data base from memory even if the data base was loaded with the LOA[D] command.

2. When installing a loadable driver in memory, the LOA[D] command searches first for a resident data base. If it finds one, it uses that and ignores the loadable version of the data base that may accompany the driver image on disk.

3. When loading a data base, LOA[D] relocates certain known pointers within the control blocks.* If the data base requires relocation of additional address pointers beyond the standard ones, it cannot be loaded with LOA[D]. It must be incorporated into the system as a resident data base by means of SYSGEN.

---

* 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.

During debugging of a loadable driver (with loadable data base), you can correct errors in the coding of the driver itself by unloading it, 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 incorporation of a user-written loadable driver as follows:

1. Creating the data base for a loadable driver.
2. Assembling a loadable driver and its data base.
3. Adding the driver and its data base to the system library.
4. Task-building a loadable driver.
5. Loading a user-written loadable driver.

3.3.1 Creating the Data Base for a Loadable Driver

In creating the data base for your loadable driver, you must decide whether the data base will be resident or loadable. If you decide upon a resident data base, you follow the procedure for creating the data base of a resident driver with the exception of initializing the device interrupt vector (see Section 3.2.1). If, however, you decide upon a loadable data base for your driver, you take the following steps:

1. 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. 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.
2. A useful convention is to name your data base source xxTAB and your driver source xxDRV where xx is the 2-character device mnemonic.
3. Place the DCB first in the assembly source. This is absolutely required. 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--for examples see the file [11,10] SYSTB.MAC created by Phase 1 of SYSGEN. 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).
4. 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 (i.e., immediately following the final word of the data base). These labels are absolutely required. xx represents the 2-character device mnemonic.

3.3.2 Assembling a Loadable Driver and Its Data Base

During SYSGEN Phase 2, you can assemble your loadable drivers at the same time that you assemble resident drivers. To do this, you would use the following command line:

```
MAC>xxDRV,xxDRV=[1,1]EXEMC/ML,[200,200]RSXMC/PA:1,xxDRV
```

If you decided upon a resident data base for your loadable driver, assembly of the data base during SYSGEN Phase 2 is described in Section 3.2.2. If, however, you are using a loadable data base for your driver (and assuming that you choose to assemble it at the same point during SYSGEN Phase 2), use the following input to MAC:

```
MAC>xxTAB,xxTAB=[1,1]EXEMC/ML,[200,200]RSXMC/PA:1,xxTAB
```

3.3.3 Adding the Driver and Its Data Base to the System Library

If you are using a resident data base for your driver, the data base is added to the Executive object module library during SYSGEN Phase 2. For loadable data bases, however, you use the following command to add both the driver and its data base to the same library:

```
LBR>RSX11M/RP = [200,200]xxDRV,xxTAB
```

3.3.4 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.

3.3.4.1 **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 filename and a symbol-definition filename as TKB output. Both files must be placed in the UFD corresponding to the system UIC that will be in effect when the LOA[D] command is issued. The filenames 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 look like this (user input underlined):

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

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

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-table file that contains definitions of Executive global symbols. Continuing the paper-tape reader driver example from Item 1 above, further TKB input might look like this:

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

   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 SYSGEN 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>[1,1]EXELIB/LB
TKB>/
```

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

6. You must direct the Task Builder not to allocate space for a stack within the driver:

```
TKB>STACK=0
```

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
TKB>//
>
```

   On mapped systems the starting address of the partition must be 120000 octal. That is, the loadable driver must be mapped to kernel APR5.

   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).

3.3.4.2 **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 UIC of some of the files ([1,50] and [1,20] instead of [1,54] and [1,24]).

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

3.3.5 Loading a User-Written Loadable Driver

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

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

LOA[D] searches first for a resident data base for the driver being loaded. If none is found, LOA[D] looks for the following global symbols in the 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.

3.4 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, should be easily debugged. This debugging process requires that you understand the following topics, each of which is discussed in a separate subsection:

1. Debugging aids and tools.
2. Fault isolation.
3. Fault tracing
4. 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 SYSGEN:

1. Executive stack and register dump
2. XDT
3. Panic dump
4. Crash dump analysis (CDA) support routine.

You need not select any of this software during SYSGEN. 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 SYSGEN, you can indicate that you want a dump of the Executive stack and the registers when a crash occurs. This dump will be provided in the following manner:

1. A system error, or the XDT X command (described in the next section), or operator manipulation of the switch register following a halt causes processing to resume at location 40(8).
2. Location 40(8) contains a JMP to 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
```

```
LOCATION CONTENTS

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 -- 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 RSX-11 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 RSX-11 ODT Manual can be used as a guide to XDT operation.

XDT may be included in a system during Phase 1 of system generation. The query:

DO YOU WANT TO INCLUDE THE EXECUTIVE DEBUGGING TOOL? [Y/N]:

is posed. If the answer is affirmative, then XDT is automatically included in the generated system. When the resultant system is bootstrapped, XDT gains 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 give a G command, passing control to the RSX-11M Executive initialization code. Whenever control reaches a breakpoint, a printout similar to that of RSX-11 ODT occurs.

A forced entry to XDT can be executed at any time from a privileged terminal by means of the MCR Breakpoint (BRK) 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 Breakpoint 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 OPEN function 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 (PANIC) saves registers R0 through R6 and then halts, awaiting dump limits to be entered.

The procedure for entering dump limits depends on whether or not your processor has a console switch register. The subsections that follow describe the procedure in each instance.

3.4.1.3.1 **Using PANIC 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(8) when the Panic Dump routine first halts. This causes dump of the system stack and the general registers. The limit 520(8) changes if the highest interrupt vector is above 400(8). 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.

3.4.1.3.2 **Using PANIC 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 PANIC 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<CR>
   $D low-address<CR>
   $L nnnnnn<CR>
   $S <CR>
   ```

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<CR>
$D high-address<CR>
$L mmmmmm<CR>
$S <CR>
```

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

3.4.1.3.3 **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 right to left, 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      ^@
```

3.4.1.4 **Crash Dump Analysis Support Routine** - The crash dump analysis (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:

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

When the system faults, you must immediately determine which of these four causes is responsible. In this section we present 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 following discussions 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.)

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

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(8), 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 may 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 appropriate PDP-11 Processor Handbook for details.

After preserving the PS and PC, invoke your resident debugging aid: enter 40(8) in the switch register, press LOAD ADDR, and then press START. The contents of 40(8) 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 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(8) as in Case 3.

This brings us to an equivalent status for the four fault situations.

3.4.2.2 **Pertinent Fault Isolation Data** - Before you attempt to locate the fault, we strongly advise you to 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, we advise you to 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:

PS

PC

The Stack

R0 through R6

Pertinent device registers

The dynamic storage region

The device tables

System common

These data represent a minimal requirement 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).

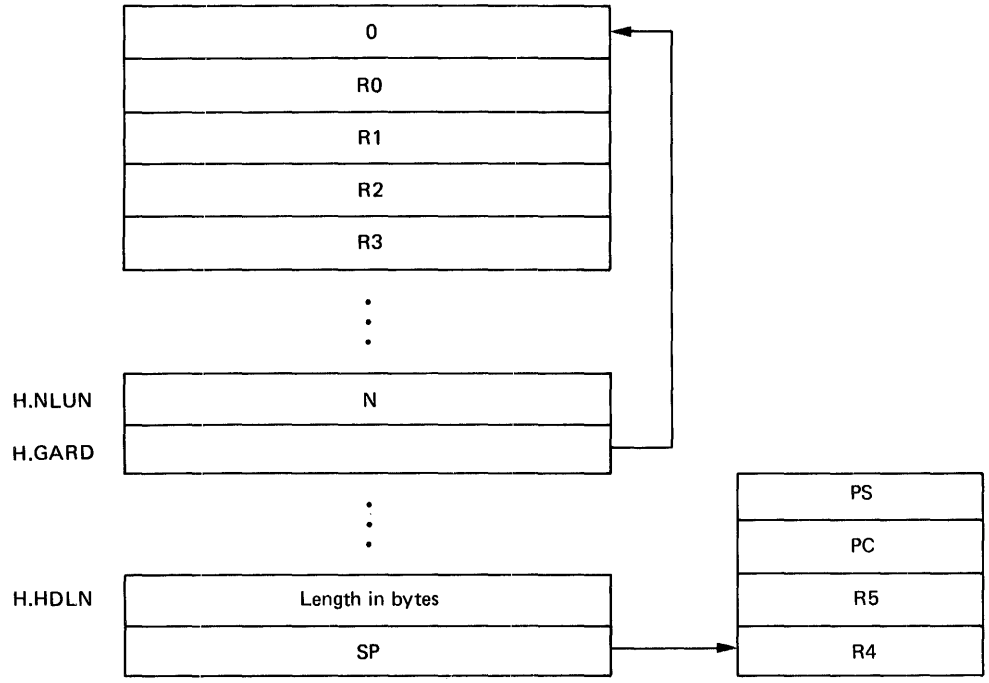


Figure 3-1 Task Header on an Unmapped System

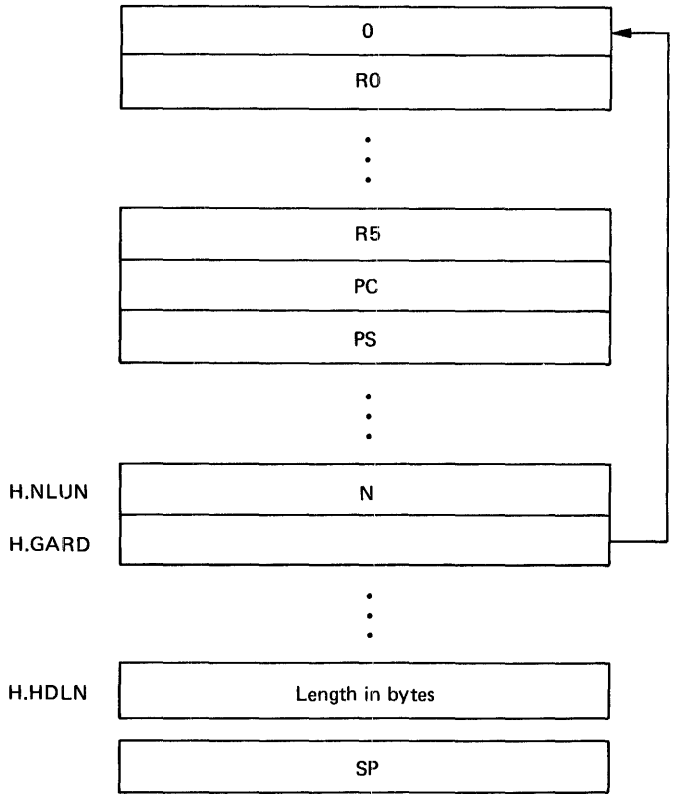


Figure 3-2 Task Header on a Mapped System

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.

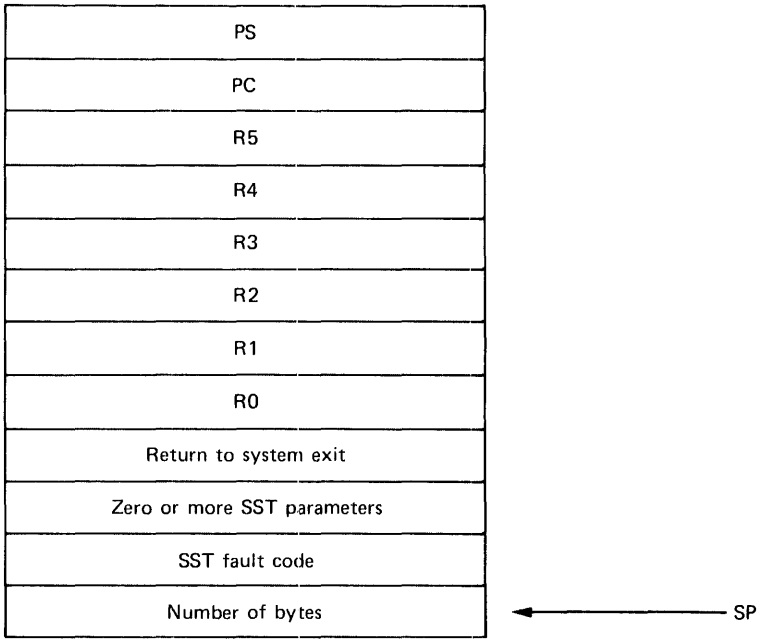


Figure 3-3 Stack Structure: Internal SST Fault

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 PS and PC are transferred. There are no SST parameters.

If the failure is detected in $DRDSP, the stack is the same as 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.

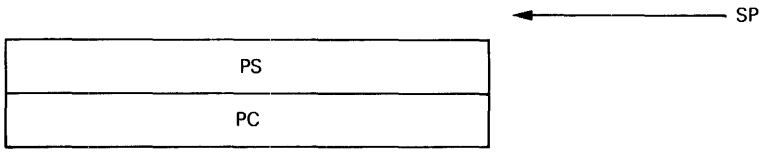


Figure 3-4 Stack Structure: Abnormal SST Fault

Abnormal SST failures occur when it is not possible to push information on 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 PS of the IOT instruction from the stack, which in this case is incorrect. Thus, the SP appears to be 4 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 Top of Stack (TOS)+4. This fault results from issuing an RTI when 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. TOS points to a word containing 40100. Suppose that location 40100 contains a halt. This indicates that the original SP was four bytes below the final SP, and fault tracing should begin from the original SP.

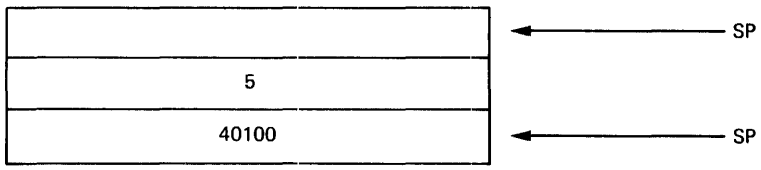


Figure 3-5 Stack Structure: Data Items on Stack

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 five steps:

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=[200,200]
>RUN $MAC                    ! OR RUN $BIGMAC
MAC>XXDRV=[1,1]EXEMC/ML,[200,200]RSXMC/PA:1,XXDRV
MAC>USRTB=[1,1]EXEMC/ML,[200,200]RSXMC/PA:1,USRTB
MAC>^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]
>RUN $LBR
LBR>RSX11M/RP=[200,200]XXDRV,USRTB
LBR>^Z
```

3. Rebuild the Executive.

   Because an updated driver is to be reinserted into the system, the Executive, of which the driver is a part, must be relinked. The following commands are an example of this relinking:

```
>SET /UIC=[1,2x]
>RUN $TKB        !  OR RUN $BIGTKB
TKB>@RSXBLD
TKB>^Z

>SET /UIC=[1,5x]
>RUN $PIP
PIP>RSX11M.SYS/NV=RSX11M.TSK
PIP>^Z
```

4. Incorporate tasks into the system using Virtual MCR.

   Run Virtual MCR (VMR) to incorporate tasks into the system. This step in the procedure requires that you:

   - Establish system partitions
   - Release all unused space to the dynamic storage region
   - Install tasks (at least FCP, INS, MOU, and MCR)
   - Exit from Virtual MCR

   The following is an example of this step:

```
>RUN $VMR/UIC=[1,5x]     ! RUN VIRTUAL MCR
ENTER FILENAME:RSX11M.SYS        ! VMR PROMPTS FOR FILE NAME
VMR>SET /MAIN=SYSPAR:1300:100:TASK ! SET UP SYSTEM PARTITION
VMR>SET /MAIN=PAR14K:400:700:TASK  ! SET UP 14K PARTITION
VMR>SET /SUB=PAR14K:GEN:400:400    ! SET UP 8K SUB PARTITION
VMR>SET /POOL=400          ! ADD FREE SPACE TO POOL
VMR>INS BOO                ! INSTALL BOOT
VMR>INS DMO                ! INSTALL DISMOUNT
VMR>INS FCPNMH             ! INSTALL FILE SYSTEM
VMR>INS IND                ! INSTALL INDIRECT FILE PROCESSOR
VMR>INS INI                ! INSTALL INITVOLUME
VMR>INS INS                ! INSTALL INSTALL
VMR>INS MCR                ! INSTALL MCR
VMR>INS MOU                ! INSTALL MOUNT
VMR>INS SAV                ! INSTALL SAVE
VMR>INS TKN                ! INSTALL TASK TERMINATION TASK
VMR>INS UFD                ! INSTALL USER FILE DIRECTORY BUILDER
VMR>^Z                     ! EXIT FROM VIRTUAL MCR
```

   The eleven INStall commands above can be placed in an indirect VMR file by Phase 2 of SYSGEN. Instead of entering each command, you could then enter, for example, the following:

```
@[200,200]INSTALL
```

5. Bootstrap the new system.

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

```
>INS BOO;-1
```

   Then issue the following command:

```
>BOO [1,5x]RSX11M
```

NOTE

If the newly created Executive is larger or smaller than the old one, the system may not run properly after exiting VMR. In this case the procedure outlined above amounts to supporting two systems on the same volume. See the RSX-11M System Generation Manual for the procedure to follow to support multiple systems on one volume.

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 command UNL, 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, and

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:

1. The I/O packet
2. The DCB
3. The UCB
4. The SCB
5. The 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, and
- 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 parameter (I.LNK) does 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.

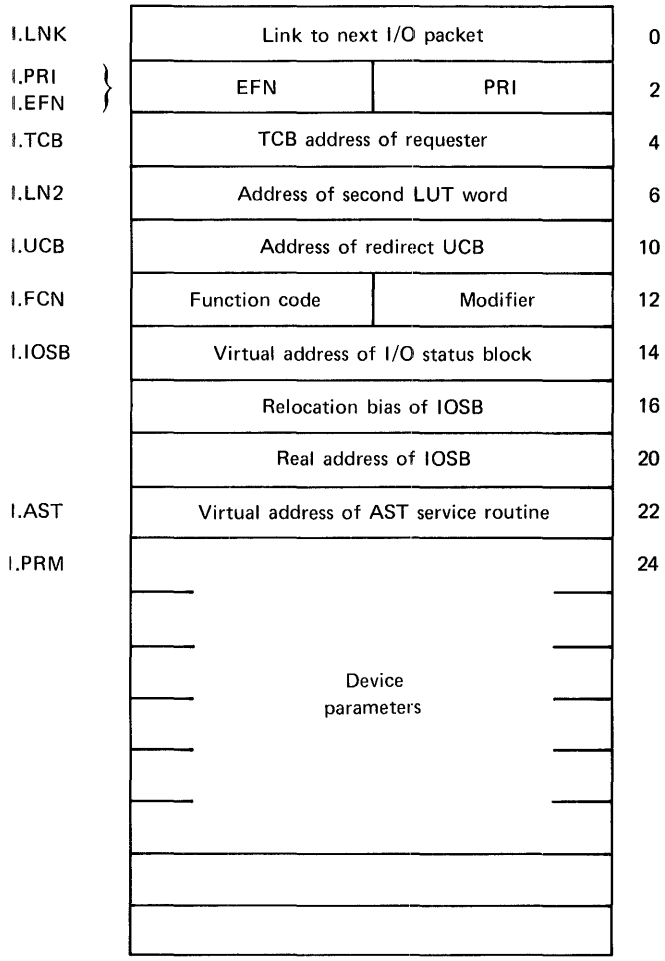


Figure 4-1 I/O Packet Format

I.PRI

Driver access:

Not referenced.

Description:

Priority copied from the TCB of the requesting task.

I.TCB

Driver access:

Not referenced.

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:

Not referenced.

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 Redirect command.

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(8), its block number is 32(8).

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(8), the DIB is equal to 10(8).

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.

We defer discussion of the address doubleword 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 6 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)+2.

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 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.).

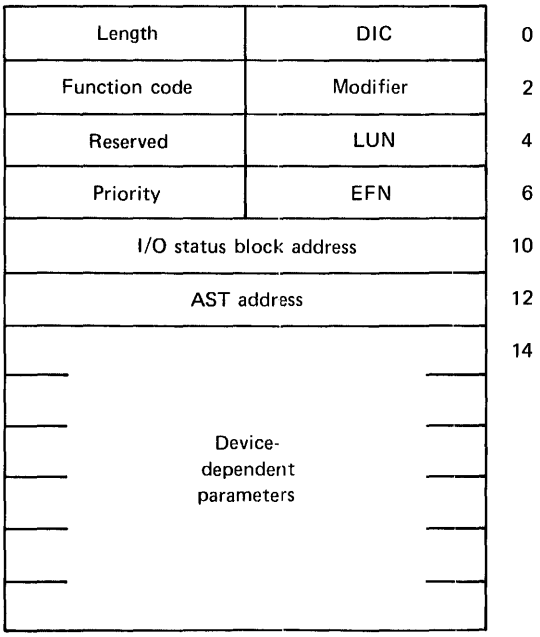


Figure 4-2 QIO Directive Parameter Block (DPB)

Modifier:

Device-dependent modifier bits.

Reserved:

Reserved byte; must not be used.

LUN (required):

Logical Unit Number.

Priority:

Request priority. Ignored by RSX-11M, but space must be allocated for RSX-11D compatibility.

EFN (optional):

Event flag number. Zero indicates no event flag.

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 = 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 must 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 the loadable-driver option has been selected.

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

D.LNK (link to next DCB)*

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.

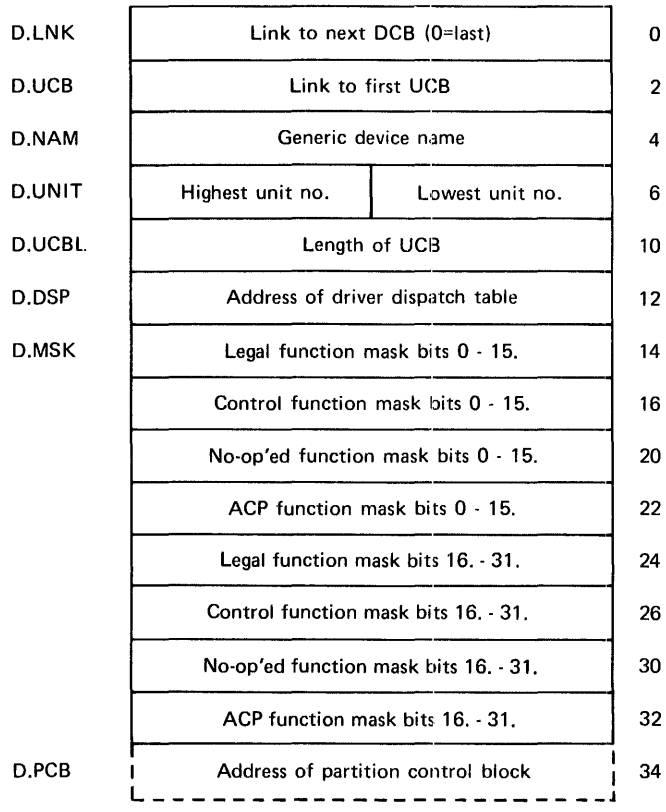


Figure 4-3 Device Control Block

* Parenthesized contents 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 UCB's for a given DCB must have the same length. The specified length must include prefix words (U.LUIC and U.OWN) if present.

D.DSP (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 it simply supplies the address of a return.

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 Timeout:

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 LOAd only for units that are online and have UC.PWF set.

D.MSK (function masks)

Driver access:

Initialized, not referenced.

Description:

There are 8 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. The following description applies only to non-file-structured devices.* Four masks, with 2 words per mask, are described by the bit configurations that you establish for these words:

1. Legal function mask
2. Control function mask
3. No-op'ed function mask
4. 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 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 exact 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'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.
```

* Although no DIGITAL publication describes writing drivers for file-structured devices (drivers that interface with F11ACP), many users have successfully written a disk/drum driver by using a DIGITAL-supplied driver as a template. For example, the RF11 driver (DFDRV) could be modified to be a drum driver.

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'ed 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, the corresponding mask bits may be set at your option. 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 no-op'ed. 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 no-op'ed.

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.) Other ACP functions that might be included fall into the non-conventional driver classification and are beyond the scope of this document.

D.PCB (Partition Control Block)

Driver access:

Initialized, not referenced.

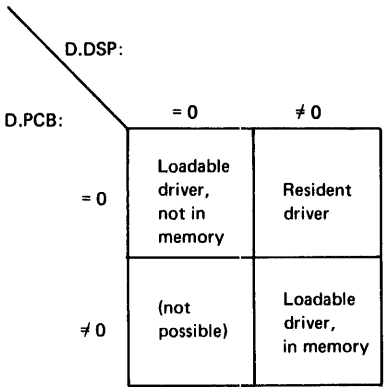
Description:

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. The DCB can be extended by adding words after D.PCB.

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

The Executive uses D.PCB together with D.DSP (the address of the driver dispatch table) to determine whether a driver is loadable or resident, and, if loadable, whether or not it is in memory. Zero and nonzero values for these two pointers have the following meanings:



4.1.2.2 **Establishing I/O Function Masks** - Table 4-1 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 4-1
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

(continued on next page)

Table 4-1 (Cont.)
Mask Values for Standard I/O Functions

Bit #	Mask Value	Related Symbolic	I/O Function
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 4-1, 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 no-op'ed, 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 4-1 must be established.

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.

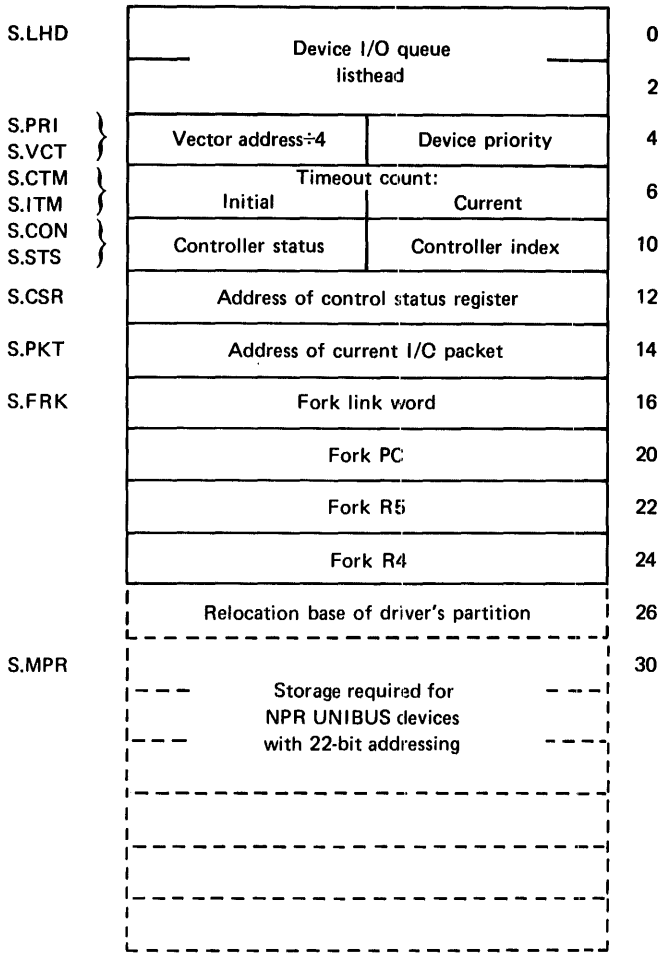


Figure 4-4 Status Control Block

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

S.LHD (first word equals zero; second word points to first)*

Driver access:

Initialized, not referenced.

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.

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

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. These symbolic values are defined 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 four)

Driver access:

Initialized, not referenced.

Description:

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

S.CTM (initialize to zero)

Driver access:

Not initialized, read-write.

Description:

RSX-11M supports device timeout, which enables a driver to limit the time that elapses between the issuing of an I/O operation and its termination. The current timeout count (in seconds) is initialized by moving S.ITM (initial timeout count) into S.CTM. The Executive clock service (in module TDSCH) examines active times, decrements them, and, if they reach 0, calls the driver at its device timeout entry point.

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

S.ITM (set to initial timeout count)

Driver access:

Initialized, read-only.

Description:

Contains the initial timeout value.

S.CON (controller number times 2)

Driver access:

Initialized, read-only.

Description:

Controller number multiplied by 2. This field is used by drivers that are written to support more than one controller. 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 offline bit at bootstrap so that system software can be interchanged between systems without an intervening system generation. The MCR LOAD function also references S.CSR when it processes a loadable data base. Otherwise, only the driver itself accesses S.CSR.

S.PKT (reserve 1 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 4 or 5 words of storage)

Driver access:

Not initialized, not referenced.

Description:

The 4 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 5 words long instead of 4 if two conditions are met:

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

2. The system is mapped.

If these conditions are met, and the fork block is 5 words long, you must not use the fork block for any other purpose. In other words, you may not 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 above conditions are met.

S.MPR (reserve 6 words of storage)

Driver access:

Initialized, read-only.

Description:

Drivers use the 6 words starting at S.MPR for non-processor request (NPR) devices attached to a PDP-11/70 with 22-bit addressing. 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:

U.LUIC

Driver access:

Not initialized, not referenced.

Description:

For terminal UCB's only, and only in multiuser systems: the logon UIC of the user at the particular terminal.

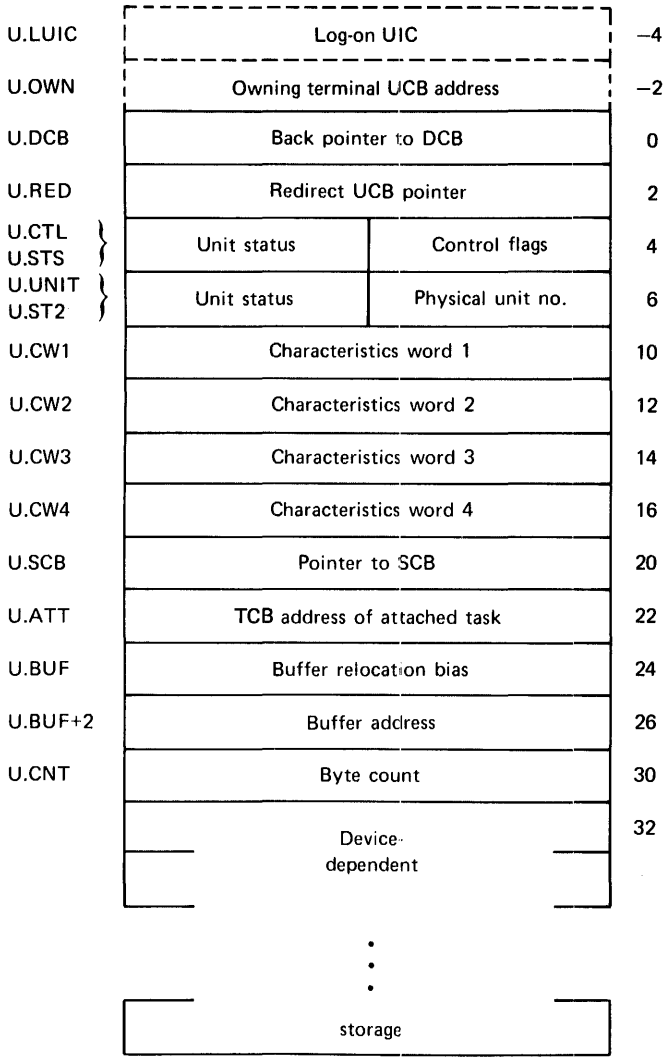


Figure 4-5 Unit Control Block

U.OWN (initialize to zero)

Driver access:

Initialized, not referenced.

Description:

Only in multiuser systems: 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 = 0, then byte alignment of data buffers is allowed. If UC.ALG = 1, then buffers must be word-aligned.

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 online, 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 online 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, 2, or 3.

00 - Any buffer modulus valid
01 - Must have word alignment modulus
10 - Combination invalid
11 - Must have double word-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. 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 offline (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.UMD  Bit  7--Device supports user-mode diagnostics
DV.MBC  Bit  8--Device attached to a 22-bit MASSBUS
                controller
DV.SWL  Bit  9--Unit is software write-locked
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).*

U.CW3 (initialize to zero)

Driver access:

Initialized, read-write.

Description:

Specific to a given device driver (available for working storage or constants).*

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.

---

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

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 2 words of storage)

Driver access:

Not initialized, read-write.

Description:

U.BUF labels 2 consecutive words that serve as a communication region between $GTPKT and the driver. If a non-transfer function is indicated (in D.MSK), then U.BUF, U.BUF+2, and U.CNT receive the first 3 parameter words from the I/O Packet.

For transfer operations, the format of these 2 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 2 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) operation 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 3 words contain the first 3 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 not dynamically. This practice makes the driver code independent of device register address assignments and of the actual location of the interrupt vector. The driver data structure must include a storage assignment and initialization for the interrupt vector with the priority set to PR7. See lines 81 thru 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 LOA[D] when the driver is loaded. When a driver is unloaded, UNLoad 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 4 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 4 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 1 word, TEMP is not needed to store the PS, and the UCB address is always the first (and only) entry in CNTBL.

```
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:

1. Single or multiple controllers
2. Loadable or resident drivers
3. Mapped or unmapped systems

All the code in Figure 4-6 between lines 7 and 26 may be replaced by 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[D] 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[D] function allocates a block of dynamic memory in kernel address space to contain the interrupt coding. This block, called the Interrupt Control Block (ICB), also contains coding to:

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

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

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

4.3.1 Format

The format of the INTSV$ macro is:

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

where:

xx — is the 2-character device mnemonic.

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

nctlr — is the number of controllers the driver services.

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

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

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

R5 is the UCB address.

Example:

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

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, SYSXT.MAC, QUEUE.MAC, CORAL.MAC, and REQSB.MAC under UIC [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 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 discussed in this chapter (Get Word and Put Word) 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 SYSGEN dialog. This requirement is spelled out in the descriptions that follow.

5.3 SERVICE CALLS

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

$ACHKB/$ACHCK

ADDRESS CHECK

These routines are in the file IOSUB. A driver may call either routine to address-check a task buffer while the task is the current task. The Address Check 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 IOSUB. It is used only for PDP-11/70 NPR devices requiring UNIBUS Mapping Registers. 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 IOSUB. It is used only for PDP-11/70 NPR devices requiring UNIBUS Mapping Registers. 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 RESERVED.
;-
```

$DVMSG

DEVICE MESSAGE OUTPUT

Device Message Output is in file IOSUB.

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
```

$FORK

FORK

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

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

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

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

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

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

$FORK1

FORK1

Fork1 is in the file SYSXT. A driver calls $FORK1 to bypass the clearing of its timeout count when it switches from a partially interruptable level to a fully interruptable level (refer also to the description of the $FORK routine).

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

Get Byte is in the file BFCTL. Get byte 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

Get Packet 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

Get Word is in the file BFCTL. It manipulates words U.BUF and U.BUF+2 in the UCB. Get Word is conditionally assembled. If a user-written driver requires this routine, the following question must be answered affirmatively in Phase 1 of SYSGEN:

IS THE EXECUTIVE ROUTINE $GTWRD REQUIRED? [Y/N]:

This question is asked only if you have answered "yes" to the question:

ARE YOU PLANNING TO INCLUDE A USER WRITTEN DRIVER? [Y/N]:

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

Interrupt Save 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

Interrupt Exit is in the file SYSXT.

Calling sequence:

```
    JMP    $INTXT
```

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 routine call $IOFIN, which destroys R4.

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

$IOFIN

I/O FINISH

I/O Finish is in the file IOSUB. Most drivers do not call I/O Finish, 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 I/O Finish 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 IOSUB. It is used only for PDP-11/70 NPR devices requiring UNIBUS Mapping Registers. 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 AN 11/70 PROCESSOR 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 IOSUB. It is used only for PDP-11/70 NPR devices that require UNIBUS Mapping Registers and support parallel operations. 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 AN 11/70 PROCESSOR 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
;
; OUTPUTS:
;
;       THE UNIBUS MAP REGISTERS NECESSARY TO EFFECT THE
;       TRANSFER ARE LOADED
;
; NOTE: REGISTER R3 IS PRESERVED ACROSS CALL.
;-
```

$PTBYT

PUT BYTE

Put Byte is in the file BFCTL. Put Byte 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

Put Word is in the file BFCTL. It manipulates words U.BUF and U.BUF+2 in the UCB. Put Word is conditionally assembled. If a user-written driver requires this routine, the following question must be answered affirmatively in Phase 1 of SYSGEN:

IS THE EXECUTIVE ROUTINE $PTWRD REQUIRED? [Y/N]:

This question is asked only if you have answered "yes" to the question:

ARE YOU PLANNING 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. Queue Insertion by Priority 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.
;-
```

$RELOC

RELOCATE

Relocate is in the file IOSUB. 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:

```
;+
; **-SRELOC-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 IOSUB. It is used only for PDP-11/70 NPR devices requiring UNIBUS Mapping Registers. 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 on to 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 IOSUB. It is used only for PDP-11/70 NPR devices that require UNIBUS Mapping Registers and support parallel operations. 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 on to the stack before returning to the caller. A driver, therefore, should not use the stack for temporary data storage when calling this routine.

CHAPTER 6

INCLUDING A USER-WRITTEN DRIVER--TWO EXAMPLES

The first example that follows is a complete illustration of the procedures required to add a resident driver and resident data base to an RSX-11M system to 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 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 SYSGEN created data structures.

6.1 DEVICE DESCRIPTION

The PC11 Paper Tape Reader/Punch is capable of reading 8-hole, unoiled, perforated paper tape at 300 characters-per-second, and punching tape at 50 characters-per-second. 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 1's and 0's. In punching tape, a mechanism translates logic levels representing 1's and 0's to the presence or absence 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, 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, in a particular case, 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
Write Logical Block
Attach Device
Detach Device
Access File For Read/Write
Access File For Read/Write/Extend
Deaccess File
Write Virtual Block

Cancel I/O is mandatory. Write Logical Block is the only transfer function actually supported.

Attach/Detach are control functions. The three Access/Deaccess functions are legal for FCS and RMS compatibility, but are no-op'ed. Write Virtual Block is legal but is converted to Write Logical Block 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(10) bit-range words to form a mask and all the 16-31(10) 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 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*
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.
47            .WORD    140000           ;NO-P'ED FUNCTION MASK CODES 0-15.
48            .WORD    0                ;ACP FUNCTION MASK CODES 0-15.
49            .WORD    5                ;LEGAL FUNCTION MASK CODES 16.-31.
50            .WORD    0                ;CONTROL FUNCTION MASK CODES 16.-31.
51            .WORD    1                ;NO-OP'ED FUNCTION MASK CODES 16.-31.
52            .WORD    4                ;ACP FUNCTION MASK CODES 16.-31.
53  ;
54  ; PAPER TAPE PUNCH UNIT CONTROL BLOCK
55  ;
56  .PP0::
57  PPST=.
58            .WORD    PPDCB            ;BACK POINTER TO DCB
59            .WORD    .-2              ;POINTER TO REDIRECT UNIT UCB
60            .BYTE    UC.ATT,0         ;CONTROL PROCESSING FLAG (PASS CONTROL
61                                      ;   ON ATTACH/DETACH), UNIT STATUS
```

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

```
                .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 classic RSX-11M form, being separated into processing code for:

Initiator

Power Failure

Interrupt

Timeout

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(10) nulls each for header and trailer.

Power Failure and Cancel I/O are handled by means of device timeout, as is the device-not-ready condition.

The PP 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 detach 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. ;
25. ; C. A. ANDERS 15-MAR-76
26. ;
27. ;         CA001 -- ADDITION OF LOADABLE DRIVER SUPPORT.
28. ;
29. ; T. J. PASCUSNIK 4-APR-76
30. ;
31. ;         TP031 -- EXECUTIVE DATA STRUCTURE CHANGES.
32. ;
33. ;
34. ; PC11 PAPER TAPE PUNCH DRIVER
35. ;
36. ; MACRO LIBRARY CALLS
37. ;
38.
39.           .MCALL  ABODF$,HWDDF$,PKTDF$,TCBDF$
40.           ABODF$                  ;DEFINE TASK ABORT CODES
41.           HWDDF$                  ;DEFINE HARDWARE REGISTER SYMBOLS
42.           PKTDF$                  ;DEFINE I/O PACKET OFFSETS
43.           TCBDF$                  ;DEFINE TASK CONTROL BLOCK OFFSETS
44.
45. ;
46. ; EQUATED SYMBOLS
47. ;
48. ; PAPER TAPE PUNCH STATUS WORD BIT DEFINITIONS (U.CW2)
49. ;
50.
```

```
51.   WAIT=100000                    ;WAITING FOR DEVICE TO COME ON-LINE (1=YES)
52.   ABORT=40000                    ;ABORT CURRENT I/O REQUEST (1=YES)
53.   TRAIL=200                      ;CURRENTLY PUNCHING TRAILER (1=YES)
54.
55. ;
56. ; LOCAL DATA
57. ;
58. ; CONTROLLER IMPURE DATA TABLES (INDEXED BY CONTROLLER NUMBER)
59. ;
60.
61.   CNTBL:  .BLKW   P$$P11          ;ADDRESS OF UNIT CONTROL BLOCK
62.
63.
64.           .IF GT  P$$P11-1
65.
66.   TEMP:   .BLKW   1               ;TEMPORARY STORAGE FOR CONTROLLER NUMBER
67.
68.           .ENDC                        ; CA001
69.                                        ; CA001
70.
71. ;
72. ; DRIVER DISPATCH TABLE
73. ;
74.
75. $PPTBL::  .WORD   PPINI           ;DEVICE INITIATOR ENTRY POINT
76.           .WORD   PPCAN           ;CANCEL I/O OPERATION ENTRY POINT
77.           .WORD   PPOUT           ;DEVICE TIMEOUT ENTRY POINT
78.           .WORD   PPPWF           ;POWERFAIL ENTRY POINT
79.
80. ;+
81. ; **-PPINI-PC11 PAPER TAPE PUNCH CONTROLLER INITIATOR
82. ;
83. ; THIS ROUTINE IS ENTERED FROM THE QUEUE I/O DIRECTIVE WHEN AN I/O REQUEST
84. ; IS QUEUED AND AT THE END OF A PREVIOUS I/O OPERATION TO PROPAGATE THE EXECU-
85. ; TION OF THE DRIVER. IF THE SPECIFIED CONTROLLER IS NOT BUSY, THEN AN ATTEMPT
86. ; IS MADE TO DEQUEUE THE NEXT I/O REQUEST. ELSE A RETURN TO THE CALLER IS
87. ; EXECUTED. IF THE DEQUEUE ATTEMPT IS SUCCESSFUL, THEN THE NEXT I/O OPER-
88. ; ATION IS INITIATED. A RETURN TO THE CALLER IS THEN EXECUTED.
89. ;
90. ; INPUTS:
91. ;
92. ; R5=ADDRESS OF THE UCB OF THE CONTROLLER TO BE INITIATED.
93. ;
94. ; OUTPUTS:
95. ;
96. ; IF THE SPECIFIED CONTROLLER IS NOT BUSY AND AN I/O REQUEST IS WAIT-
97. ; ING TO BE PROCESSED, THEN THE REQUEST IS DEQUEUED AND THE I/O OPER-
98. ; ATION IS INITIATED.
99. ;-
100.
101.          .ENABL  LSB
102.  PPINI:  CALL    $GTPKT          ;GET AN I/O PACKET TO PROCESS
103.          BCS     PPPWF           ;IF CS CONTROLLER BUSY OR NO REQUEST
104.
105. ;
106. ; THE FOLLOWING ARGUMENTS ARE RETURNED BY $GTPKT:
107. ;
108. ;         R1=ADDRESS OF THE I/O REQUEST PACKET.
109. ;         R2=PHYSICAL UNIT NUMBER OF THE REQUEST UCB.
110. ;         R3=CONTROLLER INDEX.
111. ;         R4=ADDRESS OF THE STATUS CONTROL BLOCK.
112. ;         R5=ADDRESS OF THE Uc THE CONTROLLER TO BE INITIATED.
113. ;
114. ; PAPER TAPE PUNCH I/O REQUEST PACKET FORMAT:
115. ;
116. ;         WD. 00 -- I/O QUEUE THREAD WORD.
117. ;         WD. 01 -- REQUEST PRIORITY, EVENT FLAG NUMBER.
118. ;         WD. 02 -- ADDRESS OF THE TCB OF THE REQUESTER TASK.
119. ;         WD. 03 -- POINTER TO SECOND LUN WORD IN REQUESTER TASK HEADER.
120. ;         WD. 04 -- CONTENTS OF THE FIRST LUN WORD IN REQUESTER TASK HEADER (UCB)
121. ;         WD. 05 -- I/O FUNCTION CODE (IO.WLB, IO.ATT OR IO.DET).
122. ;         WD. 06 -- VIRTUAL ADDRESS OF I/O STATUS BLOCK.
123. ;         WD. 07 -- RELOCATION BIAS OF I/O STATUS BLOCK.
```

```
124. ;          WD. 10 -- I/O STATUS BLOCK ADDRESS (REAL OR DISPLACEMENT + 140000).
125. ;          WD. 11 -- VIRTUAL ADDRESS OF AST SERVICE ROUTINE.
126. ;          WD. 12 -- RELOCATION BIAS OF I/O BUFFER.
127. ;          WD. 13 -- BUFFER ADDRESS OF I/O TRANSFER.
128. ;          WD. 14 -- NUMBER OF BYTES TO BE TRANSFERED.
129. ;          WD. 15 -- NOT USED.
130. ;          WD. 16 -- NOT USED.
131. ;          WD. 17 -- NOT USED.
132. ;          WD. 20 -- NOT USED.
133. ;
134.
135.            MOV     R5,CNTBL(R3)     ;SAVE UCB POINTER FOR INTERRUPT ROUTINE
136.            CLR     U.CW2(R5)        ;CLEAR ALL SWITCHES
137.            CMPB    I.FCN+1(R1),#IO.WLB/256. ;WRITE LOGICAL BLOCK FUNCTION?
138.            BEQ     10$              ;IF EQ YES
139.            MOV     I.TCB(R1),R0     ;GET REQUESTOR TCB ADDRESS
140.            BIT     #T2.ABO,T.ST2(R0) ;TASK BEING ABORTED?   ; TP031
141.            BNE     65$              ;IF NE YES - DON'T PUNCH TRAILER
142.            BIS     #TRAIL,U.CW2(R5) ;OTHERWISE FUNCTION IS ATTACH OR DETACH
143.                                     ;   SET FLAG TO PUNCH TRAILER
144.            MOV     #170.,U.CNT(R5) ;SET COUNT FOR 170 NULLS
145. 10$:       BIS     #WAIT,U.CW2(R5) ;ASSUME WAIT FOR DEVICE OFF LINE
146.            TST     @S.CSR(R4)       ;DEVICE OFF LINE?
147.            BMI     80$              ;IF MI YES
148. 20$:       BIC     #WAIT,U.CW2(R5) ;DEVICE ON LINE, CLEAR WAIT CONDITION
149.            MOVB    S.ITM(R4),S.CTM(R4) ;SET TIMEOUT COUNT
150.            MOV     #100,@S.CSR(R4) ;ENABLE INTERRUPTS
151.
152. ;
153. ; POWERFAIL IS HANDLED VIA THE DEVICE TIMEOUT FACILITY AND THEREFORE CAUSES
154. ; NO IMMEDIATE ACTION ON THE DEVICE. THIS IS DONE TO AVOID A RACE CONDITION
155. ; THAT COULD EXIST IN RESTARTING THE I/O OPERATION
156. ;
157.
158.   PPPWF:  RETURN                   ;
159.
160. ;+
161. ; **-$PPINT-PC11 PAPER TAPE PUNCH CONTROLLER INTERUPTS
162. ;-
163.
164. $PPINT::                           ;;;REF LABEL
165.            INTSV$  PP,PR4,P$$P11    ;;;GENERATE INTERRUPT SAVE CODE  ; CA001
166.            MOV     U.SCB(R5),R4     ;;;GET ADDRESS OF STATUS CONTROL BLOCK
167.            MOVB    S.ITM(R4),S.CTM(R4) ;;;RESET TIMEOUT COUNT
168.            MOV     S.CSR(R4),R4     ;;;POINT R4 TO CONTROL STATUS REGISTER
169.            MOV     (R4)+,U.CW3(R5) ;;;SAVE STATUS
170.            BMI     60$              ;;;IF MI, ERROR
171.            SUB     #1,U.CNT(R5)     ;;;DECREMENT CHARACTER COUNT
172.            BCS     50$              ;;;IF CS, THEN DONE
173.            TSTB    U.CW2(R5)        ;;;CURRENTLY PUNCHING TRAILER?
174.            BPL     30$              ;;;IF PL NO
175.            CLRB    (R4)             ;;;LOAD NULL INTO OUTPUT REGISTER
176.            BR      40$              ;;;BRANCH TO LOAD IT
177. 30$:       CALL    $GTBYT           ;;;GET NEXT BYTE FROM USER BUFFER
178.            MOVB    (SP)+,(R4)       ;;;LOAD BYTE INTO OUTPUT REGISTER
179. 40$:       JMP     $INTXT           ;;;EXIT FROM INTERRUPT
180. 50$:       INC     U.CNT(R5)        ;;;RESET BYTE COUNT
181. 60$:       CLR     -(R4)            ;;;DISABLE PUNCH INTERRUPTS
182.            CALL    $FORK            ;;;CREATE SYSTEM PROCESS
183.            MOV     U.SCB(R5),R4     ;POINT R4 TO SCB
184.            MOV     S.PKT(R4),R1     ;POINT R1 TO I/O PACKET
185.            MOV     I.PRM+4(R1),R1   ;   AND PICK UP CHARACTER COUNT
186.            SUB     U.CNT(R5),R1     ;CALCULATE CHARACTERS TRANSFERRED
187.            MOV     #IS.SUC&377,R0   ;ASSUME SUCCESSFUL TRANSFER
188.            TST     U.CW3(R5)        ;DEVICE ERROR?
189.            BPL     70$              ;IF PL NO
190. 65$:       MOV     #IE.VER&377,R0   ;UNRECOVERABLE HARDWARE ERROR CODE
191. 70$:       CALL    $IODON           ;INITIATE I/O COMPLETION
192.            BR      PPINI            ;BRANCH BACK FOR NEXT REQUEST
193.
194. ;
195. ; DEVICE TIMEOUT RESULTS IN A NOT READY MESSAGE BEING PUT OUT 4 TIMES A
196. ; MINUTE. TIMEOUTS ARE CAUSED BY POWERFAILURE AND PUNCH FAULT CONDITIONS.
197. ;
```

```
198.
199. PPOUT:    CLRB    @S.CSR(R4)       ;;;DISABLE PUNCH INTERRUPT
200.           CLRB    PS               ;;;ALLOW INTERRUPTS
201. 80$:      MOV     #IE.DNR&377,R0   ;ASSUME DEVICE NOT READY ERROR
202.           MOV     U.CW2(R5),R1     ;ARE WE WAITING FOR DEVICE READY?
203.           BPL     70$              ;IF PL NO, TERMINATE I/O REQUEST
204.           MOV     #IE.ABO&377,R0   ;ASSUME REQUEST IS TO BE ABORTED
205.           ASL     R1               ;ABORT REQUEST?
206.           BMI     70$              ;IF MI YES
207.           TST     @S.CSR(R4)       ;PUNCH READY?
208.           BPL     20$              ;IF PL YES
209.           MOV     #T.NDNR,R0       ;SET FOR NOT READY MESSAGE
210.           MOVB    #1,S.CTM(R4)     ;SET TIMEOUT FOR 1 SECOND
211.           DECB    S.STS(R4)        ;TIME TO OUTPUT MESSAGE?
212.           BNE     PPPWF            ;IF NE NO
213.           MOVB    #15.,S.STS(R4) ;SET TO OUTPUT NEXT MESSAGE IN 15. SECONDS
214.           CALLR   $DVMSG           ;OUTPUT MESSAGE AND RETURN
215.           .DSABL  LSB
216.
217. ;
218. ; CANCEL I/O OPERATION-FORCE I/O TO COMPLETE IF DEVICE IS NOT READY
219. ;
220.
221. PPCAN:    CMP     R1,I.TCB(R0)     ;;;REQUEST FOR CURRENT TASK?
222.           BNE     10$              ;;;IF NE NO
223.           BIS     #ABORT,U.CW2(R5) ;;;SET FOR ABORT IF DEVICE NOT READY
224. 10$:      RETURN                   ;;;
225.
226.           .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:

1. 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.)

2. The routine that is referenced as the initiator entry point in the driver dispatch table performs the following actions:

   a. Picks up the user virtual address and conditionally address-checks it.

   b. Relocates the virtual address, storing the result back into the packet.

   c. Inserts the packet into the I/O queue and falls through to the entry point AFINI, which calls $GTPKT.

3. 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

RSX-11M can be generated 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 is accomplished by using 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 2 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 Number (PAR#)

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

The relocation base contained in the PAR specified by the PAR# 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.

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

PDP-11/70 DRIVERS FOR NPR DEVICES

Special features must be built into drivers for non-MASSBUS NPR (non-processor request) devices attached to a PDP-11/70 with extended-memory support (22-bit addressing).

Non-MASSBUS NPR devices on the 11/70 must perform I/O transfers via UNIBUS Mapping Registers (UMRs) as described in the PDP-11/70 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:

1. Dynamically allocating UMRs for the duration of the data transfer, or
2. Dynamically allocating UMRs for longer periods of time, or
3. Statically allocating UMRs during system generation.

NOTE

In large systems, use of 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 6 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 6 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 conditionalized on the AND of 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 conditionalized on M$$EXT&M$$MGE.

Because $STMAP and $STMP1 push the address of routine $DQUMR+2 on to 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 conditionalized on M$$EXT&M$$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.

Through use of 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, the use of $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	

Figure B-1 Mapping Register Assignment Block

B.3 STATICALLY ALLOCATING UMRs DURING SYSTEM GENERATION

UMRs can be statically assigned during system generation. For systems with extended-memory support and memory-management unit support, the system generation procedure defines the symbol N$$UMR equal to a fixed number of UMRs, multiplied by 4, that are statically assigned to the system. Before assembling the Executive, the user can cause the static allocation of an additional number of UMRs by editing file RSXMC.MAC. The value of the symbol N$$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 2 bits of the 18-bit UNIBUS address. These 2 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 DEFINITIONS

```
        .IIF NDF S$$YDF , .NLIST

;
; COPYRIGHT (C) 1974,1976,1977
; DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
;
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A
; SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH THE
; INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE, OR
; ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE
; MADE AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH
; SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE TERMS. TITLE
; TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL TIMES REMAIN
; IN DEC.
;
; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
; EQUIPMENT CORPORATION.
;
; DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
; ITS SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.
;

        .MACRO  ABODF$,L,B

;+
; TASK ABORT CODES
;
; NOTE: S.COAD-S.CFLT ARE ALSO SST VECTOR OFFSETS
;-

S.COAD='B'0.                    ;ODD ADDRESS AND TRAPS TO 4
S.CSGF='B'2.                    ;SEGMENT FAULT
S.CBPT='B'4.                    ;BREAK POINT OR TRACE TRAP
S.CIOT='B'6.                    ;IOT INSTRUCTION
S.CILI='B'8.                    ;ILLEGAL OR RESERVED INSTRUCTION
S.CEMT='B'10.                   ;NON RSX EMT INSTRUCTION
S.CTRP='B'12.                   ;TRAP INSTRUCTION
S.CFLT='B'14.                   ;11/40 FLOATING POINT EXCEPTION
S.CSST='B'16.                   ;SST ABORT-BAD STACK
S.CAST='B'18.                   ;AST ABORT-BAD STACK
S.CABO='B'20.                   ;ABORT VIA DIRECTIVE
S.CLRF='B'22.                   ;TASK LOAD REQUEST FAILURE
S.CCRF='B'24.                   ;TASK CHECKPOINT READ FAILURE
S.IOMG='B'26.                   ;TASK EXIT WITH OUTSTANDING I/O
S.PRTY='B'28.                   ;TASK MEMORY PARITY ERROR

;
; TASK TERMINATION NOTIFICATION MESSAGE CODES
;
```

```
T.NDNR='B'0                     ;DEVICE NOT READY
T.NDSE='B'2                     ;DEVICE SELECT ERROR
T.NCWF='B'4                     ;CHECKPOINT WRITE FAILURE
T.NCRE='B'6                     ;CARD READER HARDWARE ERROR
T.NDMO='B'8.                    ;DISMOUNT COMPLETE
T.NLDN='B'12.                   ;LINK DOWN (NETWORKS)
T.NLUP='B'14.                   ;LINK UP (NETWORKS)

        .MACRO  ABODF$  X,Y
        .ENDM
        .ENDM

        .IIF NDF S$$YDF , .LIST
```

```
        .IIF NDF SSSYDF , .NLIST

;
; COPYRIGHT (C) 1974,1976,1977
; DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
;
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A
; SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH THE
; INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE, OR
; ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE
; MADE AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH
; SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE TERMS. TITLE
; TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL TIMES REMAIN
; IN DEC.
;
; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
; EQUIPMENT CORPORATION.
;
; DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
; ITS SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.
;

        .MACRO  CLKDF$,L,B

;+
; CLOCK QUEUE CONTROL BLOCK OFFSET DEFINITIONS
;
; CLOCK QUEUE CONTROL BLOCK
;
; THERE ARE FIVE TYPES OF CLOCK QUEUE CONTROL BLOCKS. EACH CONTROL BLOCK HAS
; THE SAME FORMAT IN THE FIRST FIVE WORDS AND DIFFERS IN THE REMAINING THREE.
;
; THE FOLLOWING CONTROL BLOCK TYPES ARE DEFINED:
;-

C.MRKT='B'0                     ;MARK TIME REQUEST
C.SCHD='B'2                     ;TASK REQUEST WITH PERIODIC RESCHEDULING
C.SSHT='B'4                     ;SINGLE SHOT TASK REQUEST
C.SYST='B'6                     ;SINGLE SHOT INTERNAL SYSTEM SUBROUTINE (IDENT)
C.SYTK='B'8.                    ;SINGLE SHOT INTERNAL SYSTEM SUBROUTINE (TASK)
C.CSTP='B'10.                   ;CLEAR STOP BIT (CONDITIONALIZED ON SHUFFLING)

;
; CLOCK QUEUE CONTROL BLOCK TYPE INDEPENDENT OFFSET DEFINTIONS
;

        .ASECT
.=0
C.LNK:'L' .BLKW 1               ;CLOCK QUEUE THREAD WORD
C.RQT:'L' .BLKB 1               ;REQUEST TYPE
C.EFN:'L' .BLKB 1               ;EVENT FLAG NUMBER (MARK TIME ONLY)
C.TCB:'L' .BLKW 1               ;TCB ADDRESS OR SYSTEM SUBROUTINE IDENTIFICATION
C.TIM:'L' .BLKW 2               ;ABSOLUTE TIME WHEN REQUEST COMES DUE

;
; CLOCK QUEUE CONTROL BLOCK-MARK TIME DEPENDENT OFFSET DEFINITIONS
;

.=C.TIM+4                       ;START OF DEPENDENT AREA
C.AST:'L' .BLKW 1               ;AST ADDRESS
C.SRC:'L' .BLKW 1               ;FLAG MASK WORD FOR 'BIS' SOURCE
C.DST:'L' .BLKW 1               ;ADDRESS OF 'BIS' DESTINATION

;
; CLOCK QUEUE CONTROL BLOCK-PERIODIC RESCHEDULING DEPENDENT OFFSET DEFINITIONS
;

.=C.TIM+4                       ;START OF DEPENDENT AREA
C.RSI:'L' .BLKW 2               ;RESCHEDULE INTERVAL IN CLOCK TICKS
C.UIC:'L' .BLKW 1               ;SCHEDULING UIC

;
; CLOCK QUEUE CONTROL BLOCK-SINGLE SHOT DEPENDENT OFFSET DEFINITIONS
;
```

```
.=C.TIM+4                       ;START OF DEPENDENT AREA
        .BLKW   2               ;TWO UNUSED WORDS
        .BLKW   1               ;SCHEDULING UIC

;
; CLOCK QUEUE CONTROL BLOCK-SINGLE SHOT INTERNAL SUBROUTINE OFFSET DEFINITIONS
;
; THERE ARE TWO TYPE CODES FOR THIS TYPE OF REQUEST:'L'
;
;       TYPE 6=SINGLE SHOT INTERNAL SUBROUTINE WITH A 16 BIT VALUE AS AN IDENTIFIER.
;       TYPE 8=SINGLE SHOT INTERNAL SUBROUTINE WITH A TCB ADDRESS AS AN IDENTIFIER.
;

.=C.TIM+4                       ;START OF DEPENDENT AREA
C.SUB:'L' .BLKW 1               ;SUBROUTINE ADDRESS
C.AR5:'L' .BLKW 1               ;RELOCATION BASE (FOR LOADABLE DRIVERS)
        .BLKW   1               ;ONE UNUSED WORD
C.LGTH='B'.                     ;LENGTH OF CLOCK QUEUE CONTROL BLOCK
        .PSECT

        .MACRO  CLKDF$  X,Y
        .ENDM
        .ENDM

        .IIF NDF S$$YDF , .LIST
```

```
        .IIF    NDF S$$YDF , .NLIST
;
; COPYRIGHT (C) 1974,1976,1977
; DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
;
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A
; SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH THE
; INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE, OR
; ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE
; MADE AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH
; SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE TERMS. TITLE
; TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL TIMES REMAIN
; IN DEC.
;
; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
; EQUIPMENT CORPORATION.
;
; DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
; ITS SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.
;

        .MACRO  DCBDF$,L,B

;+
;
; 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
D.LNK:'L' .BLKW 1               ;LINK TO NEXT DCB
D.UCB:'L' .BLKW 1               ;POINTER TO FIRST UNIT CONTROL BLOCK
D.NAM:'L' .BLKW 1               ;GENERIC DEVICE NAME
D.UNIT:'L' .BLKB 1              ;LOWEST UNIT NUMBER COVERED BY THIS DCB
        .BLKB   1               ;HIGHEST UNIT NUMBER COVERED BY THIS DCB
D.UCBL:'L' .BLKW 1              ;LENGTH OF EACH UNIT CONTROL BLOCK IN BYTES
D.DSP:'L' .BLKW 1               ;POINTER TO DRIVER DISPATCH TABLE
D.MSK:'L' .BLKW 1               ;LEGAL FUNCTION MASK  CODES 0-15.
        .BLKW   1               ;CONTROL FUNCTION MASK CODES 0-15.
        .BLKW   1               ;NOP'ED FUNCTION MASK CODES 0-15.
        .BLKW   1               ;ACP FUNCTION MASK CODES 0-15.
        .BLKW   1               ;LEGAL FUNCTION MASK CODES 16.-31.
        .BLKW   1               ;CONTROL FUNCTION MASK CODES 16.-31.
        .BLKW   1               ;NOP'ED FUNCTION MASK CODES 16.-31.
        .BLKW   1               ;ACP FUNCTION MASK CODES 16.-31.
D.PCB:'L' .BLKW 1               ;LOADABLE DRIVER PCB ADDRESS

        .PSECT

;+
; DRIVER DISPATCH TABLE OFFSET DEFINITIONS
;-

D.VINI='B'0                     ;DEVICE INITIATOR
D.VCAN='B'2                     ;CANCEL CURRENT I/O FUNCTION
D.VOUT='B'4                     ;DEVICE TIMEOUT
D.VPWF='B'6                     ;POWERFAIL RECOVERY

        .MACRO  DCBDF$,X,Y
        .ENDM
        .ENDM

        .IIF    NDF S$$YDF , .LIST
```

```
        .IIF    NDF,S$$YDF,.NLIST
        .TITLE  F11TBL  FILES 11 TABLE DEFINITIONS
        .IDENT  /0022/
;
; COPYRIGHT (C)   1976,  DIGITAL  EQUIPMENT  CORP.,  MAYNARD,   MASS.
;
; THIS SOFTWARE IS FURNISHED TO PURCHASER UNDER A LICENSE FOR USE
; ON A  SINGLE COMPUTER SYSTEM AND CAN BE COPIED (WITH INCLUSION
; OF DEC'S COPYRIGHT NOTICE) ONLY FOR USE IN SUCH SYSTEM, EXCEPT
; AS MAY OTHERWISE BE PROVIDED IN WRITING BY DEC.
;
; THE  INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
; NOTICE AND  SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
; EQUIPMENT CORPORATION.
;
; DEC  ASSUMES NO  RESPONSIBILITY  FOR  THE  USE  OR RELIABILITY
; OF ITS  SOFTWARE ON  EQUIPMENT WHICH  IS NOT SUPPLIED BY DEC.
;
; ELLEN R SIMICH 16-JUN-76 3-MAR-77 11M LOCK CHANGE
; ANDREW C. GOLDSTEIN  30 OCT 75  17:55
; PETER H. LIPMAN 12/27/73

        .MACRO  F11DF$
;
; VOLUME CONTROL BLOCK
;
        .ASECT
.=0
V.TRCT: .BLKW   1               ;TRANSACTION COUNT
V.IFWI: .BLKW   1               ;INDEX FILE WINDOW
        .IF     DF,R$$11D
V.STD:  .BLKW   1               ;STD OF TASK CHARGED WITH NODE
        .ENDC
V.FCB:  .BLKW   2               ;FILE CONTROL BLOCK LIST HEAD
V.IBLB: .BLKB   1               ;INDEX BIT MAP 1ST LBN HIGH BYTE
V.IBSZ: .BLKB   1               ;INDEX BIT MAP SIZE IN BLOCKS
        .BLKW   1               ;INDEX BITMAP 1ST LBN LOW BITS
V.FMAX: .BLKW   1               ;MAX NO. OF FILES ON VOLUME
V.WISZ: .BLKB   1               ;DFLT SIZE OF WINDOW IN NO. OF RTRV PTRS
                                ;VALUE IS < 128.
V.SBCL: .BLKB   1               ;STORAGE BIT MAP CLUSTER FACTOR
V.SBSZ: .BLKW   1               ;STORAGE BIT MAP SIZE IN BLOCKS
V.SBLB: .BLKB   1               ;STORAGE BIT MAP 1ST LBN HIGH BYTE
V.FIEX: .BLKB   1               ;DEFAULT FILE EXTEND SIZE
        .BLKW   1               ;STORAGE BIT MAP 1ST LBN LOW BITS
        .IF     DF,R$$11M
V.VOWN: .BLKW   1               ;VOLUME OWVER'S UIC
V.VPRO: .BLKW   1               ;VOLUME PROTECTION
V.VCHA: .BLKW   1               ;VOLUME CHARACTERISTICS
        .IFTF
V.FPRO: .BLKW   1               ;VOLUME DEFAULT FILE PROTECTION
        .IFT
V.VFSQ: .BLKW   1               ;VOLUME FILE SEQUENCE NUMBER
        .IFF
        .BLKW   1               ;NOT USED
        .ENDC
V.FRBK: .BLKB   1               ;NUMBER OF FREE BLOCKS ON VOLUME HIGH BYTE
V.LRUC: .BLKB   1               ;COUNT OF AVAILABLE LRU SLOTS IN FCB LIST
        .BLKW   1               ;NUMBER OF FREE BLOCKS ON VOLUME LOW BITS
        .IF     DF,R$$11D
V.LABL: .BLKB   12.             ;VOLUME LABEL (ASCII)
        .ENDC
V.STAT: .BLKB   1               ;VOLUME STATUS BYTE, CONTAINING THE FOLLOWING
        VC.IFW= 1               ; INDEX FILE IS WRITE ACCESSED
        VC.BMW= 2               ; STORAGE BITMAP FILE IS WRITE ACCESSED
V.FFNU: .BLKB   1               ; FIRST FREE INDEX FILE BITMAP BLOCK
V.LGTH:                         ;SIZE IN BYTES OF VCB

;
; FILE CONTROL BLOCK
;
        .ASECT
.=0
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
```

```
F.FNUM: .BLKW   1               ;FILE NUMBER
F.FSEQ: .BLKW   1               ;FILE SEQUENCE NUMBER
        .BLKB   1               ;NOT USED
F.FSQN: .BLKB   1               ;FILE SEGMENT NUMBER
F.FOWN: .BLKW   1               ;FILE OWNER'S UIC
F.FPRO: .BLKW   1               ;FILE PROTECTION CODE
F.UCHA: .BLKB   1               ;USER CONTROLLED CHARACTERISTICS
F.SCHA: .BLKB   1               ;SYSTEM CONTROLLED CHARACTERISTICS
F.HDLB: .BLKW   2               ;FILE HEADER LOGICAL BLOCK NUMBER
                                ;BEGINNING OF STATISTICS BLOCK
F.LBN:  .BLKW   2               ;LBN OF VIRTUAL BLOCK 1 IF CONTIGUOUS
                                ;0 IF NON CONTIGUOUS
F.SIZE: .BLKW   2               ;SIZE OF FILE IN BLOCKS
F.NACS: .BLKB   1               ;NO. OF ACCESSES
F.NLCK: .BLKB   1               ;NO. OF LOCKS
        S.STBK=.-F.LBN          ;SIZE OF STATICS BLOCK
F.STAT:                         ;FCB STATUS WORD

F.NWAC: .BLKB   1               ;NUMBER OF WRITE ACCESSORS
        .BLKB   1               ;STATUS BITS FOR FCB CONSISTING OF
        FC.WAC=100000           ;SET IF FILE ACCESSED FOR WRITE
        FC.DIR=40000            ;SET IF FCB IS IN DIRECTORY LRU
        FC.CEF=20000            ;SET IF DIRECTORY EOF NEEDS UPDATING
        FC.FCO=10000            ;SET IF TRYING TO FORCE DIRECTORY CONTIG
F.DREF: .BLKW   1               ;DIRECTORY EOF BLOCK NUMBER
F.DRNM: .BLKW   1               ;1ST WORD OF DIRECTORY NAME
        .IF     DF,R$$11M
F.FEXT: .BLKW   1               ;POINTER TO EXTENSION FCB
        .ENDC
F.FVBN: .BLKW   2               ;STARTING VBN OF THIS FILE SEGMENT
F.LKL:  .BLKW   1               ;POINTER TO LOCKED BLOCK LIST FOR FILE
F.LGTH:                         ;SIZE IN BYTES OF FCB

;
; WINDOW
;
        .ASECT
.=0
W.CTL:  .BLKW   1               ;LOW BYTE = # OF MAP ENTRIES ACTIVE
                                ;HIGH BYTE CONSISTS OF THE FOLLOWING BITS
        WI.RDV=400              ;READ VIRTUAL BLOCK ALLOWED IF SET
        WI.WRV=1000             ;WRITE VIRTUAL BLOCK ALLOWED IF SET
        WI.EXT=2000             ;EXTEND ALLOWED IF SET
        WI.LCK=4000             ;SET IF LOCKED AGAINST SHARED ACCESS
        WI.DLK=10000            ;SET IF DEACCESS LOCK ENABLED
        WI.EXL=40000            ;SET IF MANUAL UNLOCK DESIRED
        WI.BPS=100000           ;BYPASS ACCESS INTERLOCK IF SET
        .IF     DF,R$$11M
W.VBN:  .BLKB   1               ;HIGH BYTE OF 1ST VBN MAPPED BY WINDOW
W.WISZ: .BLKB   1               ;SIZE IN RTRV PTRS OF WINDOW (7 BITS)
        .BLKW   1               ;LOW ORDER WORD OF 1ST VBN MAPPED
W.FCB:  .BLKW   1               ;FILE CONTROL BLOCK ADDRESS
        .ENDC
        .IF     DF,R$$11D
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
        .ENDC
W.LKL:  .BLKW   1               ;POINTER TO LIST OF USERS LOCKED BLOCKS
W.RTRV:                         ;OFFSET TO 1ST RETRIEVAL POINTER IN WINDOW

;
; LOCKED BLOCK LIST NODE
;
        .ASECT
.=0
L.LNK:  .BLKW   1               ;LINK TO NEXT NODE IN LIST
L.WI1:  .BLKW   1               ;POINTER TO WINDOW FOR FIRST ENTRY
        .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
```

```
L.VB1:  .BLKB   1               ;HIGH ORDER VBN BYTE
L.CNT:  .BLKB   1               ;COUNT FOR ENTRY
        .BLKW   1
        .ENDC
L.LGTH:
        .PSECT
        .MACRO  F11DF$
        .ENDM   F11DF$
        .ENDM   F11DF$
        .IIF    NDF,S$$YDF,.LIST
```

```
        .IIF NDF S$$YDF , .NLIST

;
; COPYRIGHT (C) 1974,1976,1977
; DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
;
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A
; SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH THE
; INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE, OR
; ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE
; MADE AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH
; SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE TERMS. TITLE
; TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL TIMES REMAIN
; IN DEC.
;
; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
; EQUIPMENT CORPORATION.
;
; DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
; ITS SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.
;

        .MACRO  HDRDF$,L,B

;+
; TASK HEADER OFFSET DEFINITIONS
;-

        .ASECT
.=0
H.CSP:'L'.BLKW  1               ;CURRENT STACK POINTER
H.HDLN:'L'.BLKW 1               ;HEADER LENGTH IN BYTES
H.EFLM:'L'.BLKW 2               ;EVENT FLAG MASK WORD AND ADDRESS
H.CUIC:'L'.BLKW 1               ;CURRENT TASK UIC
H.DUIC:'L'.BLKW 1               ;DEFAULT TASK UIC
H.IPS:'L'.BLKW  1               ;INITIAL PROCESSOR STATUS WORD (PS)
H.IPC:'L'.BLKW  1               ;INITIAL PROGRAM COUNTER (PC)
H.ISP:'L'.BLKW  1               ;INITIAL STACK POINTER (SP)
H.ODVA:'L'.BLKW 1               ;ODT SST VECTOR ADDRESS
H.ODVL:'L'.BLKW 1               ;ODT SST VECTOR LENGTH
H.TKVA:'L'.BLKW 1               ;TASK SST VECTOR ADDRESS
H.TKVL:'L'.BLKW 1               ;TASK SST VECTOR LENGTH
H.PFVA:'L'.BLKW 1               ;POWER FAIL AST CONTROL BLOCK ADDRESS
H.FPVA:'L'.BLKW 1               ;FLOATING POINT AST CONTROL BLOCK ADDRESS
H.RCVA:'L'.BLKW 1               ;RECIEVE AST CONTROL BLOCK ADDRESS
H.EFSV:'L'.BLKW 1               ;EVENT FLAG ADDRESS SAVE ADDRESS
H.FPSA:'L'.BLKW 1               ;POINTER TO FLOATING POINT/EAE SAVE AREA
H.WND:'L'.BLKW  1               ;POINTER TO NUMBER OF WINDOW BLOCKS
H.DSW:'L'.BLKW  1               ;TASK DIRECTIVE STATUS WORD
H.FCS:'L'.BLKW  1               ;FCS IMPURE POINTER
H.FORT:'L'.BLKW 1               ;FORTRAN IMPURE POINTER
H.OVLY:'L'.BLKW 1               ;OVERLAY IMPURE POINTER
H.VEXT:'L'.BLKW 1               ;WORK AREA EXTENSION VECTOR POINTER
H.SPRI:'L'.BLKB 1               ;PRIORITY DIFFERENCE FOR SWAPPING
H.NML:'L'.BLKB  1               ;NETWORK MAILBOX LUN
H.RRVA:'L'.BLKW 1               ;RECEIVE BY REFERENCE AST CONTROL BLOCK ADDRESS
        .BLKW   3               ;RESERVED WORDS
H.GARD:'L'.BLKW 1               ;POINTER TO HEADER GUARD WORD
H.NLUN:'L'.BLKW 1               ;NUMBER OF LUN'S
H.LUN:'L'.BLKW  2               ;START OF LOGICAL UNIT TABLE

;+
; WINDOW BLOCK OFFSETS
;-

.=0
W.BPCB:'L'.BLKW 1               ;PARTITION CONTROL BLOCK ADDRESS
W.BLVR:'L'.BLKW 1               ;LOW VIRTUAL ADDRESS LIMIT
W.BHVR:'L'.BLKW 1               ;HIGH VIRTUAL ADDRESS LIMIT
W.BATT:'L'.BLKW 1               ;ADDRESS OF ATTACHMENT DESCRIPTOR
W.BSIZ:'L'.BLKW 1               ;SIZE OF WINDOW IN 32W BLOCKS
W.BOFF:'L'.BLKW 1               ;PHYSICAL MEMORY OFFSET IN 32W BLOCKS
W.BFPD:'L'.BLKB 1               ;FIRST PDR ADDRESS
W.BNPD:'L'.BLKB 1               ;NUMBER OF PDR'S TO MAP
```

```
W.BLPD:'L'.BLKW 1               ;CONTENTS OF LAST PDR
W.BLGH:'L'                      ;LENGTH OF WINDOW DESCRIPTOR
        .PSECT

        .MACRO  HDRDF$  X,Y
        .ENDM
        .ENDM

        .IIF NDF S$$YDF , .LIST
```

```
        .IIF NDF S$$YDF , .NLIST

;
; COPYRIGHT (C) 1974,1976,1977
; DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
;
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A
; SINGLE COMPUTER SYSTEM AND MAY  BE  COPIED   ONLY  WITH  THE
; INCLUSION OF THE ABOVE COPYRIGHT NOTICE.  THIS SOFTWARE,  OR
; ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED  OR  OTHERWISE
; MADE AVAILABLE TO ANY OTHER PERSON   EXCEPT FOR  USE ON SUCH
; SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE  TERMS.  TITLE
; TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL  TIMES  REMAIN
; IN DEC.
;
; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
; EQUIPMENT CORPORATION.
;
; DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
; ITS SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.
;

        .MACRO HWDDF$,L,B

;+
; HARDWARE REGISTER ADDRESSES AND STATUS CODES
;-

MPCSR='B'177746                 ;ADDRESS OF PDP-11/70 MEMORY PARITY REGISTER
MPAR='B'172100                  ;ADDRESS OF FIRST MEMORY PARITY REGISTER
PIRQ='B'177772                  ;PROGRAMMED INTERRUPT REQUEST REGISTER
PR0='B'0                        ;PROCESSOR PRIORITY 0
PR1='B'40                       ;PROCESSOR PRIORITY 1
PR4='B'200                      ;PROCESSOR PRIORITY 4
PR5='B'240                      ;PROCESSOR PRIORITY 5
PR6='B'300                      ;PROCESSOR PRIORITY 6
PR7='B'340                      ;PROCESSOR PRIORITY 7
PS='B'177776                    ;PROCESSOR STATUS WORD
SWR='B'177570                   ;CONSOLE SWITCH AND DISPLAY REGISTER
TPS='B'177564                   ;CONSOLE TERMINAL PRINTER STATUS REGISTER

;+
; EXTENDED ARITHMETIC ELEMENT REGISTERS
;-


        .IF DF  E$$EAE

AC='B'177302                    ;ACCUMULATOR
MQ='B'177304                    ;MULTIPLIER-QUOTIENT
SC='B'177310                    ;SHIFT COUNT

        .ENDC


;+
; MEMORY MANAGEMENT HARDWARE REGISTERS AND STATUS CODES
;-

        .IF DF  M$$MGE

KDSAR0='B'172360                ;KERNEL D PAR 0
KDSDR0='B'172320                ;KERNEL D PDR 0
KISAR0='B'172340                ;KERNEL I PAR 0
KISAR5='B'172352                ;KERNEL I PAR 5
KISAR6='B'172354                ;KERNEL I PAR 6
KISAR7='B'172356                ;KERNEL I PAR 7
KISDR0='B'172300                ;KERNEL I PDR 0
KISDR6='B'172314                ;KERNEL I PDR 6
KISDR7='B'172316                ;KERNEL I PAR 7
SISDR0='B'172200                ;SUPERVISOR I PDR 0
UDSAR0='B'177660                ;USER D PAR 0
UDSDR0='B'177620                ;USER D PDR 0
UISAR0='B'177640                ;USER I PAR 0
UISAR4='B'177650                ;USER I PAR 4
UISAR5='B'177652                ;USER I PAR 5
```

```
UISAR6='B'177654                ;USER I PAR 6
UISAR7='B'177656                ;USER I PAR 7
UISDR0='B'177600                ;USER I PDR 0
UISDR4='B'177610                ;USER I PDR 4
UISDR5='B'177612                ;USER I PDR 5
UISDR6='B'177614                ;USER I PDR 6
UISDR7='B'177616                ;USER I PDR 7
UBMPR='B'170200                 ;UNIBUS MAPPING REGISTER 0
CMODE='B'140000                 ;CURRENT MODE FIELD OF PS WORD
PMODE='B'30000                  ;PREVIOUS MODE FIELD OF PS WORD
SR0='B'177572                   ;SEGMENT STATUS REGISTER 0
SR3='B'172516                   ;SEGMENT STATUS REGISTER 3

        .ENDC


;+
; FEATURE SYMBOL DEFINITIONS
;-

FE.EXT='B'1                     ;11/70 EXTENDED MEMORY SUPPORT
FE.MUP='B'2                     ;MULTI-USER PROTECTION SUPPORT
FE.EXV='B'4                     ;EXECUTIVE IS SUPPORTED TO 20K
FE.DRV='B'10                    ;LOADABLE DRIVER SUPPORT
FE.PLA='B'20                    ;PLAS SUPPORT
FE.CAL='B'40                    ;DYNAMIC CHECKPOINT SPACE ALLOCATION
FE.PKT='B'100                   ;PREALLOCATION OF I/O PACKETS
FE.EXP='B'200                   ;EXTEND TASK DIRECTIVE SUPPORTED
FE.LSI='B'400                   ;PROCESSOR IS AN LSI-11
FE.CEX='B'20000                 ;COM EXEC IS LOADED
FE.MXT='B'40000                 ;MCR EXIT AFTER EACH COMMAND MODE
FE.NLG='B'100000                ;LOGINS DISABLED - MULTI-USER SUPPORT

        .MACRO  HWDDF$  X,Y
        .ENDM
        .ENDM

        .IIF NDF S$$YDF , .LIST
```

```
        .IIF    NDF S$$YDF , .NLIST
;
; COPYRIGHT (C) 1974,1976,1977
; DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
;
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A
; SINGLE COMPUTER SYSTEM AND MAY  BE  COPIED   ONLY  WITH  THE
; INCLUSION OF THE ABOVE COPYRIGHT NOTICE.  THIS SOFTWARE,  OR
; ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED  OR  OTHERWISE
; MADE AVAILABLE TO ANY OTHER PERSON   EXCEPT FOR  USE ON SUCH
; SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE  TERMS.  TITLE
; TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL  TIMES  REMAIN
; IN DEC.
;
; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
; EQUIPMENT CORPORATION.
;
; DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
; ITS SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.
;

        .MACRO  LCBDF$,L,B
;+
; 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
L.LNK:'L' .BLKW 1               ;LINK TO NEXT LCB
L.NAM:'L' .BLKW 1               ;LOGICAL NAME OF DEVICE
L.UNIT:'L' .BLKB 1              ;LOGICAL UNIT NUMBER
L.TYPE:'L' .BLKB 1              ;TYPE OF ENTRY (0=SYSTEM WIDE)
L.UCB:'L' .BLKW 1               ;TI UCB ADDRESS
L.ASG:'L' .BLKW 1               ;ASSIGNMENT UCB ADDRESS
L.LGTH='B'.-L.LNK               ;LENGTH OF LCB
        .PSECT

        .MACRO  LCBDF$,X,Y
        .ENDM
        .ENDM

        .IIF    NDF S$$YDF , .LIST
```

```
        .IIF NDF S$$YDF , .NLIST

;
; COPYRIGHT (C) 1974,1976,1977
; DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
;
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A
; SINGLE COMPUTER SYSTEM AND MAY  BE  COPIED   ONLY  WITH  THE
; INCLUSION OF THE ABOVE COPYRIGHT NOTICE.  THIS SOFTWARE,  OR
; ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED  OR  OTHERWISE
; MADE AVAILABLE TO ANY OTHER PERSON   EXCEPT FOR  USE ON SUCH
; SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE  TERMS.  TITLE
; TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL  TIMES  REMAIN
; IN DEC.
;
; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
; EQUIPMENT CORPORATION.
;
; DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
; ITS SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.
;

        .MACRO  PCBDF$  L,B,SYSDEF

;+
; PARTITION CONTROL BLOCK OFFSET DEFINITIONS
;-

        .ASECT
.=0
P.LNK:'L'.BLKW  1               ;LINK TO NEXT PARTITION PCB
P.PRI:'L'.BLKB  1               ;PRIORITY OF PARTITION
P.IOC:'L'.BLKB  1               ;I/O + I/O STATUS BLOCK COUNT
P.NAM:'L'.BLKW  2               ;PARTITION NAME IN RAD50
P.SUB:'L'.BLKW  1               ;POINTER TO NEXT SUBPARTITION
P.MAIN:'L'.BLKW 1               ;POINTER TO MAIN PARTITION


        .IF NB  SYSDEF


        .IF NDF M$$MGE

P.HDR:'L'                       ;POINTER TO HEADER CONTROL BLOCK

        .ENDC


        .IFTF

P.REL:'L'.BLKW  1               ;STARTING PHYSICAL ADDRESS OF PARTITION
P.BLKS:'L'
P.SIZE:'L'.BLKW 1               ;SIZE OF PARTITION IN BYTES
P.WAIT:'L'.BLKW 1               ;PARTITION WAIT QUEUE LISTHEAD (2 WORDS)
P.SWSZ:'L'.BLKW 1               ;PARTITION SWAP SIZE (SYSTEM ONLY)
P.BUSY:'L'.BLKB 2               ;PARTITION BUSY FLAGS
P.OWN:'L'
P.TCB:'L'.BLKW  1               ;TCB ADDRESS OF OWNER TASK
P.STAT:'L'.BLKW 1               ;PARTITION STATUS FLAGS

        .IFT


        .IF DF  M$$MGE

P.HDR:'L' .BLKW 1               ;POINTER TO HEADER CONTROL BLOCK

        .ENDC


P.PRO:'L' .BLKW 1               ;PROTECTION WORD [DEWR,DEWR,DEWR,DEWR]
P.ATT:'L' .BLKW 2               ;ATTACHMENT DESCRIPTOR LISTHEAD


        .IF NDF P$$LAS
```

```
P.LGTH='B'P.PRO                 ;LENGTH OF PARTITION CONTROL BLOCK

        .IFF

P.LGTH='B'.                     ;LENGTH OF PARTITION CONTROL BLOCK

        .ENDC


        .IFF

        .PSECT

;+
; PARTITION STATUS WORD BIT DEFINITIONS
;-

PS.OUT='B'100000                ;PARTITION IS OUT OF MEMORY(1=YES)
PS.CKP='B'40000                 ;PARTITION CHECKPOINT IN PROGRESS (1=YES)
PS.CKR='B'20000                 ;PARTITION CHECKPOINT IS REQUESTED (1=YES)
PS.CHK='B'10000                 ;PARTITION IS NOT CHECKPOINTABLE (1=YES)
PS.FXD='B'4000                  ;PARTITION IS FIXED (1=YES)
PS.PER='B'2000                  ;PARITY ERROR IN PARTITION (1=YES)
PS.LIO='B'1000                  ;MARKED BY SHUFFLER FOR LONG I/O (1=YES)
PS.NSF='B'400                   ;PARTITION IS NOT SHUFFLEABLE (1=YES)
PS.COM='B'200                   ;LIBRARY OR COMMON BLOCK (1=YES)
PS.PIC='B'100                   ;POSITION INDEPENDENT LIBRARY OR COMMON (1=YES)
PS.SYS='B'40                    ;SYSTEM CONTROLLED PARTITION (1=YES)
PS.DRV='B'20                    ;DRIVER IS LOADED IN PARTITION (1=YES)
PS.DEL='B'10                    ;PARTITION SHOULD BE DELETED WHEN NOT ATTACHED (1=YES)
PS.APR='B'7                     ;STARTING APR NUMBER MASK

;+
; ATTACHMENT DESCRIPTOR OFFSETS
;-

        .ASECT
.=0
A.PCBL:'L'.BLKW 1               ;PCB ATTACHMENT QUEUE THREAD WORD
A.PRI:'L'.BLKB  1               ;PRIORITY OF ATTACHED TASK
A.IOC:'L'.BLKB  1               ;I/O COUNT THROUGH THIS DESCRIPTOR
A.TCB:'L'.BLKW  1               ;TCB ADDRESS OF ATTACHED TASK
A.TCBL:'L'.BLKW 1               ;TCB ATTACHMENT QUEUE THREAD WORD
A.STAT:'L'.BLKB 1               ;STATUS BYTE
A.MPCT:'L'.BLKB 1               ;MAPPING COUNT OF TASK THRU THIS DESCRIPTOR
A.PCB:'L'.BLKW  1               ;PCB ADDRESS OF ATTACHED TASK
A.LGTH='B'.                     ;LENGTH OF ATTACHMENT DESCRIPTOR

;+
; ATTACHMENT DESCRIPTOR STATUS BYTE BIT DEFINITIONS
;-

        .PSECT
AS.DEL='B'10                    ;TASK HAS DELETE ACCESS (1=YES)
AS.EXT='B'4                     ;TASK HAS EXTEND ACCESS (1=YES)
AS.WRT='B'2                     ;TASK HAS WRITE ACCESS (1=YES)
AS.RED='B'1                     ;TASK HAS READ ACCESS (1=YES)

        .ENDC


        .MACRO  PCBDF$  X,Y,Z
        .ENDM
        .ENDM

        .IIF NDF S$$YDF , .LIST
```

```
        .IIF NDF S$$YDF , .NLIST

;
; COPYRIGHT (C) 1974, 1977
; DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
;
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A
; SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH THE
; INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE, OR
; ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE
; MADE AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH
; SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE TERMS. TITLE
; TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL TIMES REMAIN
; IN DEC.
;
; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
; EQUIPMENT CORPORATION.
;
; DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
; ITS SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.
;

        .MACRO  PKTDF$,L,B,SYSDEF

;+
; 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
A.KSR5:'L' .BLKW 1              ;SUBROUTINE KISAR5 BIAS (A.CBL=0)
A.DQSR:'L' .BLKW 1              ;DEQUEUE SUBROUTINE ADDRESS (A.CBL=0)
        .BLKW   1               ;AST QUEUE THREAD WORD
A.CBL:'L' .BLKW 1               ;LENGTH OF CONTROL BLOCK IN BYTES
A.BYT:'L' .BLKW 1               ;NUMBER OF BYTES TO ALLOCATE ON TASK STACK
A.AST:'L' .BLKW 1               ;AST TRAP ADDRESS
A.NPR:'L' .BLKW 1               ;NUMBER OF AST PARAMETERS
A.PRM:'L' .BLKW 1               ;FIRST AST PARAMETER

;+
; I/O PACKET OFFSET DEFINITIONS
;-

        .ASECT
.=0
I.LNK:'L' .BLKW 1               ;I/O QUEUE THREAD WORD
I.PRI:'L' .BLKB 1               ;REQUEST PRIORITY
I.EFN:'L' .BLKB 1               ;EVENT FLAG NUMBER
I.TCB:'L' .BLKW 1               ;TCB ADDRESS OF REQUESTOR
I.LN2:'L' .BLKW 1               ;POINTER TO SECOND LUN WORD
I.UCB:'L' .BLKW 1               ;POINTER TO UNIT CONTROL BLOCK
I.FCN:'L' .BLKW 1               ;I/O FUNCTION CODE
I.IOSB:'L' .BLKW        1               ;VIRTUAL ADDRESS OF I/O STATUS BLOCK
        .BLKW   1               ;I/O STATUS BLOCK RELOCATON BIAS
        .BLKW   1               ;I/O STATUS BLOCK ADDRESS
I.AST:'L' .BLKW 1               ;AST SERVICE ROUTINE ADDRESS
I.PRM:'L' .BLKW 1               ;RESERVED FOR MAPPING PARAMETER #1
        .BLKW   6               ;PARAMETERS 1 TO 6
        .BLKW   1               ;USER MODE DIAGNOSTIC PARAMETER WORD
I.ATTL='B'.                     ;MINIMUM LENGTH OF I/O PACKET (USED BY
                                ;FILE SYSTEM TO CALCULATE MAXIMUM
                                ;NUMBER OF ATTRIBUTES)
I.LGTH='B'.                     ;LENGTH OF I/O REQUEST CONTROL BLOCK

        .PSECT

        .MACRO  PKTDF$  X,Y,Z
        .ENDM
        .ENDM

        .IIF NDF S$$YDF , .LIST
```

```
        .IIF    NDF S$$YDF , .NLIST

;
; COPYRIGHT (C) 1974,1976,1977
; DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
;
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A
; SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH THE
; INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE, OR
; ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE
; MADE AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH
; SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE TERMS. TITLE
; TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL TIMES REMAIN
; IN DEC.
;
; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
; EQUIPMENT CORPORATION.
;
; DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
; ITS SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.
;

        .MACRO  SCBDF$,L,B,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 TELE-
; TYPE INTERFACED VIA A DL11-A WOULD HAVE A SCB SINCE EACH DL11-A IS AN IN-
; DEPENDENT 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
S.RCNT:'L' .BLKB 1              ;NUMBER OF REGISTERS TO COPY ON ERROR
S.ROFF:'L' .BLKB 1              ;OFFSET TO FIRST DEVICE REGISTER
S.BMSV:'L' .BLKW 1              ;SAVED I/O ACTIVE BITMAP AND POINTER TO EMB
S.BMSK:'L' .BLKW 1              ;DEVICE I/O ACTIVE BIT MASK
S.LHD:'L' .BLKW 2               ;CONTROLLER I/O QUEUE LISTHEAD
S.PRI:'L' .BLKB 1               ;DEVICE PRIORITY
S.VCT:'L' .BLKB 1               ;INTERRUPT VECTOR ADDRESS /4
S.CTM:'L' .BLKB 1               ;CURRENT TIMEOUT COUNT
S.ITM:'L' .BLKB 1               ;INITIAL TIMEOUT COUNT
S.CON:'L' .BLKB 1               ;CONTROLLER INDEX
S.STS:'L' .BLKB 1               ;CONTROLLER STATUS (0=IDLE,1=BUSY)
S.CSR:'L' .BLKW 1               ;ADDRESS OF CONTROL STATUS REGISTER
S.PKT:'L' .BLKW 1               ;ADDRESS OF CURRENT I/O PACKET
S.FRK:'L' .BLKW 1               ;FORK BLOCK LINK WORD
        .BLKW   1               ;FORK-PC
        .BLKW   1               ;FORK-R5
        .BLKW   1               ;FORK-R4


        .IF NB  SYSDEF


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

        .BLKW   1               ;FORK-DRIVER RELOCATION BASE

        .ENDC


S.CCB:'L'                       ;MIXED MASSBUS CHANNEL CONTROL BLOCK
S.MPR:'L' .BLKW 6               ;11/70 EXTENDED MEMORY UNIBUS DEVICE C-BLOCK

        .IFF

        .PSECT
```

```
;+
; STATUS CONTROL BLOCK PRIORITY BYTE CONDITION CODE STATUS BIT DEFINITIONS
;-

SP.EIP='B'1                     ;ERROR IN PROGRESS (1=YES)
SP.ENB='B'2                     ;ERROR LOGGING ENABLED (0=YES)
SP.LOG='B'4                     ;ERROR LOGGING AVAILABLE (1=YES)
SPARE=10                        ;SPARE BIT

;+
; MAPPING ASSIGNMENT BLOCK (FOR UNIBUS MAPPING REGISTER ASSIGNMENT)
;-

        .ASECT
.=0
M.LNK:'L'  .BLKW 1              ;LINK WORD
M.UMRA:'L' .BLKW 1              ;ADDRESS OF FIRST ASSIGNED UMR
M.UMRN:'L' .BLKW 1              ;NUMBER OF UMR'S ASSIGNED * 4
M.UMVL:'L' .BLKW 1              ;LOW 16 BITS MAPPED BY 1ST ASSIGNED UMR
M.UMVH:'L' .BLKB 1              ;HIGH 2 BITS MAPPED IN BITS 4 AND 5
M.BFVH:'L' .BLKB 1              ;HIGH 6 BITS OF PHYSICAL BUFFER ADDRESS
M.BFVL:'L' .BLKW 1              ;LOW 16 BITS OF PHYSICAL BUFFER ADDRESS
M.LGTH='B'.                     ;LENGTH OF MAPPING ASSIGNMENT BLOCK

        .ENDC

        .PSECT

        .MACRO  SCBDFS,X,Y,Z
        .ENDM
        .ENDM

        .IIF    NDF SSSYDF , .LIST
```

```
        .IIF NDF S$$YDF , .NLIST

;
; COPYRIGHT (C) 1974,1976,1977
; DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
;
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A
; SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH THE
; INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE, OR
; ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE
; MADE AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH
; SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE TERMS. TITLE
; TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL TIMES REMAIN
; IN DEC.
;
; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
; EQUIPMENT CORPORATION.
;
; DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
; ITS SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.
;

        .MACRO  TCBDF$,L,B,SYSDEF

;+
; TASK CONTROL BLOCK OFFSET AND STATUS DEFINITIONS
;
; TASK CONTROL BLOCK
;-

        .ASECT
.=0
T.LNK:'L' .BLKW 1               ;UTILITY LINK WORD
T.PRI:'L' .BLKB 1               ;TASK PRIORITY
T.IOC:'L' .BLKB 1               ;I/O PENDING COUNT
T.CPCB:'L' .BLKW 1              ;POINTER TO CHECKPOINT PCB
T.NAM:'L' .BLKW 2               ;TASK NAME IN RAD50
T.RCVL:'L' .BLKW 2              ;RECEIVE QUEUE LISTHEAD
T.ASTL:'L' .BLKW 2              ;AST QUEUE LISTHEAD
T.EFLG:'L' .BLKW 2              ;TASK LOCAL EVENT FLAGS 1-32
T.UCB:'L' .BLKW 1               ;UCB ADDRESS FOR PSEUDO DEVICE 'TI'
T.TCBL:'L' .BLKW 1              ;TASK LIST THREAD WORD
T.STAT:'L' .BLKW 1              ;FIRST STATUS WORD (BLOCKING BITS)
T.ST2:'L' .BLKW 1               ;SECOND STATUS WORD (STATE BITS)
T.ST3:'L' .BLKW 1               ;THIRD STATUS WORD (ATTRIBUTE BITS)
T.DPRI:'L' .BLKB 1              ;TASK'S DEFAULT PRIORITY
T.LBN:'L' .BLKB 3               ;LBN OF TASK LOAD IMAGE
T.LDV:'L' .BLKW 1               ;UCB ADDRESS OF LOAD DEVICE
T.PCB:'L' .BLKW 1               ;PCB ADDRESS OF TASK PARTITION
T.MXSZ:'L' .BLKW 1              ;MAXIMUM SIZE OF TASK IMAGE (MAPPED ONLY)
T.ACTL:'L' .BLKW 1              ;ADDRESS OF NEXT TASK IN ACTIVE LIST
T.ATT:'L' .BLKW 2               ;ATTACHMENT DESCRIPTOR LISTHEAD
T.OFF:'L' .BLKW 1               ;OFFSET TO TASK IMAGE IN PARTITION
        .BLKB 1                 ;RESERVED
T.SRCT:'L' .BLKB 1              ;SREF WITH EFN COUNT IN ALL RECEIVE QUEUES
T.RRFL:'L' .BLKW 2              ;RECEIVE BY REFERENCE LISTHEAD

        .IF NB  SYSDEF
        .IF NDF P$$LAS

T.LGTH='B'T.ATT

        .IFF

T.LGTH='B'.                     ;LENGTH OF TASK CONTROL BLOCK

        .ENDC

T.EXT='B'0                      ;LENGTH OF TCB EXTENSION

        .IFF
```

```
;+
; TASK STATUS DEFINITIONS
;
; FIRST STATUS WORD (BLOCKING BITS)
;-

TS.EXE='B'100000                ;TASK NOT IN EXECUTION (1=YES)
TS.RDN='B'40000                 ;I/O RUN DOWN IN PROGRESS (1=YES)
TS.MSG='B'20000                 ;ABORT MESSAGE BEING OUTPUT (1=YES)
TS.NRP='B'10000                 ;TASK MAPPED TO NONRESIDENT PARTITION (1=YES)
TS.RUN='B'4000                  ;TASK IS RUNNING ON ANOTHER PROCESSOR (1=YES)
TS.OUT='B'400                   ;TASK IS OUT OF MEMORY (1=YES)
TS.CKP='B'200                   ;TASK IS BEING CHECKPOINTED (1=YES)
TS.CKR='B'100                   ;TASK CHECKPOINT REQUESTED (1=YES)

;+
; TASK BLOCKING STATUS MASK
;-

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

;+
; SECOND STATUS WORD (STATE BITS)
;-

T2.AST='B'100000                ;AST IN PROGRESS (1=YES)
T2.DST='B'40000                 ;AST RECOGNITION DISABLED (1=YES)
T2.CHK='B'20000                 ;TASK NOT CHECKPOINTABLE (1=YES)
T2.CKD='B'10000                 ;CHECKPOINTING DISABLED (1=YES)
T2.BFX='B'4000                  ;TASK BEING FIXED IN MEMORY (1=YES)
T2.FXD='B'2000                  ;TASK FIXED IN MEMORY (1=YES)
T2.TIO='B'1000                  ;TASK IS ENGAGED IN TERMINAL I/O
T2.CAF='B'400                   ;DYN CHECKPOINT SPACE ALLOCATION FAILURE
T2.HLT='B'200                   ;TASK IS BEING HALTED (1=YES)
T2.ABO='B'100                   ;TASK MARKED FOR ABORT (1=YES)
T2.STP='B'40                    ;TASK STOPPED (1=YES)
T2.STP='B'20                    ;TASK STOPPED (1=YES)
T2.SPN='B'10                    ;SAVED TS.SPN ON AST IN PROGRESS
T2.SPN='B'4                     ;TASK SUSPENDED (1=YES)
T2.WFR='B'2                     ;SAVED TS.WFR ON AST IN PROGRESS
T2.WFR='B'1                     ;TASK IN WAITFOR STATE (1=YES)

;+
; THIRD STATUS WORD (ATTRIBUTE BITS)
;-

T3.ACP='B'100000                ;ANCILLARY CONTROL PROCESSOR (1=YES)
T3.PMD='B'40000                 ;DUMP TASK ON SYNCHRONOUS ABORT (0=YES)
T3.REM='B'20000                 ;REMOVE TASK ON EXIT (1=YES)
T3.PRV='B'10000                 ;TASK IS PRIVILEGED (1=YES)
T3.MCR='B'4000                  ;TASK REQUESTED AS EXTERNAL MCR FUNCTION (1=YES)
T3.SLV='B'2000                  ;TASK IS A SLAVE TASK (1=YES)
T3.CLI='B'1000                  ;TASK IS A COMMAND LINE INTERPRETER (1=YES)
T3.RST='B'400                   ;TASK IS RESTRICTED (1=YES)
T3.NSD='B'200                   ;TASK DOES NOT ALLOW SEND DATA
T3.CAL='B'100                   ;TASK HAS CHECKPOINT SPACE IN TASK IMAGE
T3.ROV='B'40                    ;TASK HAS RESIDENT OVERLAYS
T3.NET='B'20                    ;NETWORK PROTOCOL LEVEL

        .ENDC


        .PSECT
        .MACRO  TCBDF$  X,Y,Z
        .ENDM
        .ENDM

        .IIF NDF S$$YDF , .LIST
```

```
        .IIF    NDF S$$YDF , .NLIST

;
; COPYRIGHT (C) 1974,1976,1977
; DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
;
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A
; SINGLE COMPUTER SYSTEM AND MAY  BE  COPIED   ONLY  WITH  THE
; INCLUSION OF THE ABOVE COPYRIGHT NOTICE.  THIS SOFTWARE,  OR
; ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED  OR  OTHERWISE
; MADE AVAILABLE TO ANY OTHER PERSON   EXCEPT FOR  USE ON SUCH
; SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE  TERMS.  TITLE
; TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL  TIMES  REMAIN
; IN DEC.
;
; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
; EQUIPMENT CORPORATION.
;
; DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
; ITS SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.
;

        .MACRO  UCBDF$,L,B

;+
; 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
.=177774
U.LUIC:'L' .BLKW 1              ;LOGIN UIC - MULTI USER SYSTEMS ONLY
U.OWN:'L' .BLKW 1               ;OWNING TERMINAL - MULTI USER SYSTEMS ONLY
U.DCB:'L' .BLKW 1               ;BACK POINTER TO DCB
U.RED:'L' .BLKW 1               ;POINTER TO REDIRECT UNIT UCB
U.CTL:'L' .BLKB 1               ;CONTROL PROCESSING FLAGS
U.STS:'L' .BLKB 1               ;UNIT STATUS
U.UNIT:'L' .BLKB 1              ;PHYSICAL UNIT NUMBER
U.ST2:'L' .BLKB 1               ;UNIT STATUS EXTENSION
U.CW1:'L' .BLKW 1               ;FIRST DEVICE CHARACTERISTICS WORD
U.CW2:'L' .BLKW 1               ;SECOND DEVICE CHARACTERISTICS WORD
U.CW3:'L' .BLKW 1               ;THIRD DEVICE CHARACTERISTICS WORD
U.CW4:'L' .BLKW 1               ;FOURTH DEVICE CHARACTERISTICS WORD
U.SCB:'L' .BLKW 1               ;POINTER TO SCB
U.ATT:'L' .BLKW 1               ;TCB ADDRESS OF ATTACHED TASK
U.BUF:'L' .BLKW 1               ;RELOCATION BIAS OF CURRENT I/O REQUEST
        .BLKW   1               ;BUFFER ADDRESS OF CURRENT I/O REQUEST
U.CNT:'L' .BLKW 1               ;BYTE COUNT OF CURRENT I/O REQUEST
U.ACP='B'U.CNT+2                ;ADDRESS OF TCB OF MOUNTED ACP
U.VCB='B'U.CNT+4                ;ADDRESS OF VOLUME CONTROL BLOCK
U.CBF='B'U.CNT+2                ;CONTROL BUFFER RELOCATION AND ADDRESS
U.UIC='B'U.CNT+<9.*2>           ;TERMINAL UIC (TERMINALS ONLY)

        .PSECT

;+
; DEVICE TABLE STATUS DEFINITIONS
;
; DEVICE CHARACTERISTICS WORD 1 (U.CW1) DEVICE TYPE DEFINITION BITS.
;-

DV.REC='B'1                     ;RECORD ORIENTED DEVICE (1=YES)
DV.CCL='B'2                     ;CARRIAGE CONTROL DEVICE (1=YES)
DV.TTY='B'4                     ;TERMINAL DEVICE (1=YES)
DV.DIR='B'10                    ;FILE STRUCTURED DEVICE (1=YES)
DV.SDI='B'20                    ;SINGLE DIRECTORY DEVICE (1=YES)
DV.SQD='B'40                    ;SEQUENTIAL DEVICE (1=YES)
DV.MXD='B'100                   ;MASS BUS DEVICE (1=YES)
```

```
DV.UMD='B'200                   ;USER MODE DIAGNOSTICS SUPPORTED (1=YES)
DV.SWL='B'1000                  ;UNIT SOFTWARE WRITE LOCKED (1=YES)
DV.ISP='B'2000                  ;INPUT SPOOLED DEVICE (1=YES)
DV.OSP='B'4000                  ;OUTPUT SPOOLED DEVICE (1=YES)
DV.PSE='B'10000                 ;PSEUDO DEVICE (1=YES)
DV.COM='B'20000                 ;DEVICE IS MOUNTABLE AS COM CHANNEL (1=YES)
DV.F11='B'40000                 ;DEVICE IS MOUNTABLE AS F11 DEVICE (1=YES)
DV.MNT='B'100000                ;DEVICE IS MOUNTABLE (1=YES)

;+
; TERMINAL DEPENDENT CHARACTERISTICS WORD 2 (U.CW2) BIT DEFINITIONS
;-

U2.DH1='B'100000                ;UNIT IS A MULTIPLEXER (1=YES)
U2.DJ1='B'40000                 ;UNIT IS A DJ11 (1=YES)
U2.RMT='B'20000                 ;UNIT IS REMOTE (1=YES)
U2.L8S='B'10000                 ;UNIT IS LA180S (1=YES)
U2.NEC='B'4000                  ;DON'T ECHO SOLICITED INPUT (1=YES)
U2.CRT='B'2000                  ;UNIT IS A CRT (1=YES)
U2.ESC='B'1000                  ;UNIT GENERATES ESCAPE SEQUENCES (1=YES)
U2.LOG='B'400                   ;USER LOGGED ON TERMINAL (0=YES)
U2.SLV='B'200                   ;UNIT IS A SLAVE TERMINAL (1=YES)
U2.DZ1='B'100                   ;UNIT IS A DZ11 (1=YES)
U2.HLD='B'40                    ;TERMINAL IS IN HOLD SCREEN MODE (1=YES)
U2.AT.='B'20                    ;MCR COMMAND AT. BEING PROCESSED (1=YES)
U2.PRV='B'10                    ;UNIT IS A PRIVILEGED TERMINAL (1=YES)
U2.L3S='B'4                     ;UNIT IS A LA30S TERMINAL (1=YES)
U2.VT5='B'2                     ;UNIT IS A VT05B TERMINAL (1=YES)
U2.LWC='B'1                     ;LOWER CASE TO UPPER CASE CONVERSION (1=YES)

;+
; RH11-RS03/RS04 CHARACTERISTICS WORD 2 (U.CW2) BIT DEFINITIONS
;-

U2.R04='B'100000                ;UNIT IS A RS04 (1=YES)

;+
; RH11-TU16 CHARACTERISTICS WORD 2 (U.CW2) BIT DEFINITIONS
;-

U2.7CH='B'10000                 ;UNIT IS A 7 CHANNEL DRIVE (1=YES)

;+
; UNIT CONTROL PROCESSING FLAG DEFINITIONS
;-

UC.ALG='B'200                   ;BYTE ALIGNMENT ALLOWED (1=NO)
UC.NPR='B'100                   ;DEVICE IS AN NPR DEVICE (1=YES)
UC.QUE='B'40                    ;CALL DRIVER BEFORE QUEUING (1=YES)
UC.PWF='B'20                    ;CALL DRIVER AT POWERFAIL ALWAYS (1=YES)
UC.ATT='B'10                    ;CALL DRIVER ON ATTACH/DETACH (1=YES)
UC.KIL='B'4                     ;CALL DRIVER AT I/O KILL ALWAYS (1=YES)
UC.LGH='B'3                     ;TRANSFER LENGTH MASK BITS

;+
; UNIT STATUS BIT DEFINTIONS
;-

US.BSY='B'200                   ;UNIT IS BUSY (1=YES)
US.MNT='B'100                   ;UNIT IS MOUNTED (0=YES)
US.FOR='B'40                    ;UNIT IS MOUNTED AS FOREIGN VOLUME (1=YES)
US.MDM='B'20                    ;UNIT IS MARKED FOR DISMOUNT (1=YES)

;+
; CARD READER DEPENDENT UNIT STATUS BIT DEFINITIONS
;-

US.ABO='B'1                     ;UNIT IS MARKED FOR ABORT IF NOT READY (1=YES)
US.MDE='B'2                     ;UNIT IS IN 029 TRANSLATION NODE (1=YES)

;+
; FILES-11 DEPENDENT UNIT STATUS BITS
;-

US.WCK='B'10                    ;WRITE CHECK ENABLED (1=YES)
US.SPU='B'2                     ;UNIT IS SPINNING UP (1=YES)
```

```
;+
; TERMINAL DEPENDENT UNIT STATUS BIT DEFINITIONS
;-

US.DSB='B'10                    ;UNIT IS DISABLED (1=YES)
US.CRW='B'4                     ;UNIT IS WAITING FOR CARRIER (1=YES)
US.ECH='B'2                     ;UNIT HAS ECHO IN PROGRESS (1=YES)
US.OUT='B'1                     ;UNIT IS EXPECTING OUTPUT INTERRUPT (1=YES)

;+
; LPS11 DEPENDENT UNIT STATUS BIT DEFINITIONS
;-

US.FRK='B'2                     ;FORK IN PROGRESS (1=YES)
US.SHR='B'1                     ;SHAREABLE FUNCTION IN PROGRESS (0='B'YES)

;+
; ANSI MAGTAPE DEPENDANT UNIT STATUS BITS
;-
US.LAB='B'4                     ; UNIT HAS LABELED TAPE ON IT (1=YES)

;+
; UNIT STATUS EXTENSION (U.ST2) BIT DEFINITIONS
;-

US.OFL='B'1                     ;UNIT OFFLINE (1=YES)
US.RED='B'2                     ;UNIT REDIRECTABLE (0=YES)
US.PUB='B'4                     ;UNIT IS PUBLIC DEVICE (1=YES)
US.UMD='B'10                    ;UNIT ATTACHED FOR DIAGNOSTICS (1=YES)
        .MACRO  UCBDF$,X,Y
        .ENDM
        .ENDM

        .IIF    NDF S$$YDF , .NLIST
```

INDEX

Update No.1
RSX-11M
Guide to Writing an I/O Driver
AA-2600D-TC, AD-2600D-T1

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 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

Please cut along this line.

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

RT/C SOFTWARE PUBLICATIONS TW/A14
DIGITAL EQUIPMENT CORPORATION
1925 ANDOVER STREET
TEWKSBURY, MASSACHUSETTS 01876

Do Not Tear - Fold Here

Cut Along Dotted Line

UPDATE NOTICE NO.1

RSX-11M Guide to Writing an I/O Driver
AD-2600D-T1

May 1979

Insert this Update Notice page in the manual as a means of maintaining an up-to-date record of changes to the manual.

NEW AND CHANGED INFORMATION

This update reflects software changes and additions made in RSX-11M Version 3.2.

Copyright © 1979 Digital Equipment Corporation

INSTRUCTIONS

Place the following pages in the RSX-11M Guide to Writing an I/O Driver as replacements for, or additions to, current pages. The changes made on replacement pages are indicated in the outside margin by change bars (▮) for additions, and bullets (•) for deletions.

Old Page	New Page
Title Page/Copyright Page	Title Page/Copyright Page
iii/iv through vii/viii	iii/iv through vii/viii
2-13/2-14	2-13/2-14
3-3/3-4	3-3/3-4
3-9/3-10	3-9/3-10
4-9/4-10	4-9/4-10
4-11/4-12	4-11/4-12
-	4-12.1/blank
4-13/4-14	4-13/4-14
4-15/4-16	4-15/4-16
4-19/4-20 through 4-23/4-24	4-19/4-20 through 4-23/4-24
5-15/5-16	5-15/5-16
-	5-18.1/blank
5-23/5-24	5-23/5-24
6-3/6-4	6-3/6-4
B-1/B-2	B-1/B-2
B-3/B-4	B-3/B-4
Index-1/Index-2	Index-1/Index-2
Index-3/Index-4	Index-3/Index-4
Reader's Comments	Reader's Comments

Additional copies of this update to the RSX-11M Guide to Writing an I/O Driver may be ordered from the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754. Order Code: AD-2600D-T1. The order code of the base manual is AA-2600D-TC.