RSX-11M

Guide to Writing an I/O Driver

Order No. DEC-11-OMWDA-B-D

digital

RSX-11M

Guide to Writing an I/O Driver

Order No. DEC-11-OMWDA-B-D

RSX-11M Version 2

digital equipment corporation · maynard. massachusetts

First Printing, April 1975
Revised, September 1975

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 to the terms of such license.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by Digital.

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		

LIMITED RIGHTS LEGEND

Contract No. _____

Contractor or Subcontractor: Digital Equipment Corporation

All the material contained herein is considered limited rights data under such contract.

CONTENTS

CONTENTS (Cont.)

CONTENTS (Cont.)

FIGURES

TABLES

PREFACE

0.1 MANUAL OBJECTIVES AND READER ASSUMPTIONS

The goal of this manual is to provide all the information necessary to successfully 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 non-conventional. Complexity of device servicing requirements sets the dividing line, a line not easily established in go, no-go terms.

The manual assumes the reader fully understands the device for which he is writing a driver, and has complete familiarity with the PDP-11 computer, its peripheral devices, and the software supplied with an RSX-11M system. Complete familiarity implies an in-depth exposure to the following RSX-11M manuals (see section 0.3 below):

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

Although this manual is organized tutorially, our reader class assumptions require a system programmer level of expertise; thus, the manual will not contain definitions of data processing terms and concepts familiar to senior level professionals.

As adjuncts to this manual, the reader is 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 the source code contained in the files IOSUB, SYSXT, DRQIO, BFCTL, and DRDSP (stored under UIC [11,10] on the source disk) should prove beneficial.

0.2 STRUCTURE OF THE DOCUMENT

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

Chapter 1 is a functional description of the RSX-11M device level I/O system, covering both data structure and code requirements.

Chapter 2 details how a user-written driver is incorporated into the system.

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

Chapter 3 provides programming level details on I/O data structures.

Chapter 4 covers all the I/O-related Executive services.

Chapter 5 is an example of a user-written driver.

Appendix A describes the Address Doubleword.

Appendix B lists system macros which supply symbolic offsets for system data structures.

0.3 ASSOCIATED DOCUMENTS

Other manuals closely allied to the purposes of this document are described briefly in the RSX-11M/RSX-11S Documentation Directory, Order No. DEC-11-OMUGA-B-D. 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.

CHAPTER 1

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

1.1 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, a substantial effort has been expended in the design of RSX-11M's data structures. These structures are used to drive the centralized routines; the effect is to substantially reduce the size of individual I/O drivers. The table structures, of course, 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, has enabled RSX-11M to meet its original memory and performance goals.

In a DEC-released system, DEC-supported drivers are included into the user-tailored system via system generation queries. User-written drivers require the user to create object files for I/O data structures and driver code. These object files are built and incorporated during the generation of the user-tailored system.

1.2 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.

1.2.1 I/O Hierarchy

The RSX-11M I/O system is structured as a loose hierarchy. The term "loose" simply indicates that the hierarchy can be entered at any of its levels; this characteristic is contrasted to "tight" hierarchies which permit entry only from the top. Tight hierarchies exist principally for security and protection, but are costly in their consumption of equipment resources. Figure 1-1 shows the loose I/O system hierarchy.

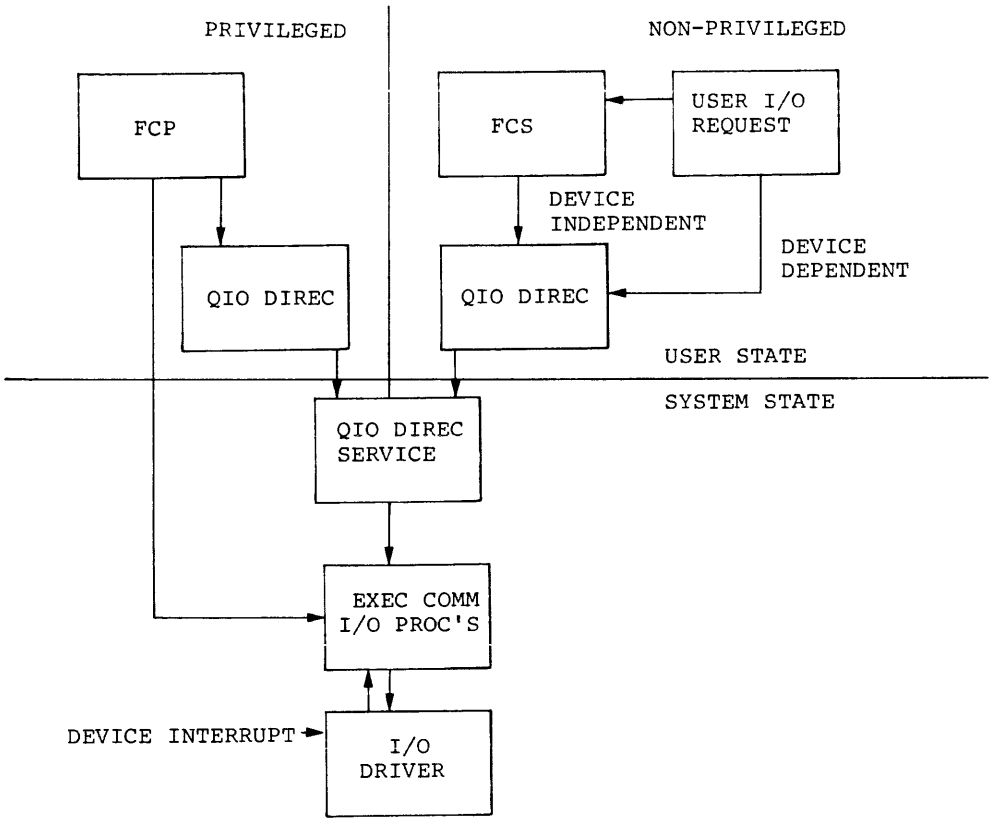

Figure 1-1
I/O Control Flow

1.2.1.1 FCS - At the top of the hierarchy is File Control Services (FCS) which provides device-independent access to devices included in a given system configuration. The user task has two levels with which to interface with FCS; 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.

The discussion of FCS has been purposely terse because its existence is completely transparent to the driver and rarely concerns the writer of a conventional driver. FCS will accept a user request, interpret it, and perform all operations necessary to carry out the user request.

1.2.1.2 QIO - The QIO directive is the lowest level of task I/O. Any task may issue a QIO directive. The QIO directive allows more direct control over devices which are connected to a system and for which an I/O driver exists. The QIO directive forces all I/O requests from user task's to go through the Executive. The Executive prevents tasks from destructively interfering with each other and with the Executive.

1.2.1.3 Executive I/O Processing - The processing of I/O requests by the Executive I/O system is accomplished via:

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 a privileged task; it 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. No direct connection exists between FCP and a driver.

Logical blocks are 256 words in length; it is the responsibility of the driver to convert a logical block number into a physical address on a file-structured device.

Within the system, FCP exists as a privileged task, possessing all the attributes of privileged tasks. FCP requires a partition in which to execute. Drivers, on the other hand, are specialized, permanently-resident 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 will be examined in considerable detail later. Executive subroutine services and QIO directive processing are shown as distinct components but are, in fact, both parts of the Executive. These are the routines which centralize common driver functions, thus eliminating duplicate code sequences among drivers.

The description of the I/O hierarchy and interrelationships is now sufficiently complete to allow a more direct consideration of the role fulfilled by an I/O driver in an RSX-11M system.

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

Every driver has five entry points:

1. Device interrupt*;
2. I/O initiator;
3. Device timeout;
4. Cancel I/O, and
5. Power failure.

The first entry point is entered via a hardware interrupt; the last four are entered by calls from the Executive. These entry points are descriptive enough in and of themselves to provide direct insight into the responsibilities of a driver; the functional details follow.

1.2.2.1 Device Interrupt - Control is passed to this entry point when a device previously initiated by the driver has completed an I/O operation and has caused an interrupt in the central processor. The connection to the driver in this instance is direct; the Executive is not involved.

1.2.2.2 I/O Initiator - This entry point is called by the Executive to inform the driver that work for it is waiting to be done. In effect, this is a wakeup signal for the driver.

1.2.2.3 Device Timeout - When a driver initiates an I/O operation, it establishes a timeout count. If the function does not complete within the specified time interval, the Executive will note the time lapse and call the driver at this entry point.

1.2.2.4 Cancel I/O - A number of circumstances arise within the system which require that a driver terminate an in-progress I/O operation. When this 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.

1.2.2.5 Power Failure - Two conditions can initiate a call to the driver when power is restored following a power failure. First, the power failure entry point is automatically called by the Executive any time the controller is busy. Secondly, 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. If power fails, and the conditions

---

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

exist for power failure code initiation, the Executive will so inform the driver by calling it at the power failure entry point when power is restored. Frequently, a driver's response to a power failure or 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, since recovery will eventually be effected via the timeout entry point.

Also, when the system is bootstrapped, a power failure interrupt is simulated. This simulation gives drivers the opportunity to carry out any pre-operational initialization deemed appropriate.

1.2.2.6 Summary - Role of an I/O Driver - Functionally, the driver in RSX-11M has responsibility for:

1. Servicing device interrupts;
2. Initiating I/O operations;
3. Cancelling in-progress I/O operations, and
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 via Executive calls. A driver never receives a QIO request directly, and has no direct interaction with FCP.

At this point, a functional description of the role of an I/O driver in RSX-11M has been presented. In the next three sections, the

Data structures,

Executive services, and

Programming conventions and protocol

related to I/O drivers will be discussed. The chapter closes with a section discussing the flow of an I/O request, from the issuance of a QIO directive to the delivery of the requested data to the task. Data structure interrelationships are also covered.

1.3 DATA STRUCTURES

There are seven data structures with which an I/O driver interacts:

1. Device Control Blocks (DCB's);
2. Unit Control Blocks (UCB's);
3. Status Control Blocks (SCB's);
4. The I/O Packet;
5. The I/O Queue;

6. The Fork List, and

7. Device Interrupt Vectors.

The first four are most important to the driver, since it is via 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 properly understand the complete interaction of an I/O driver with the Executive, their role in the system will also be described. The I/O Queue is a list of I/O Packets which are built by the QIO directive, principally from information in the caller's Directive Parameter Block (DPB).

Entry to a driver following a device interrupt is direct via the appropriate hardware device interrupt vector. Since the driver writer is responsible for properly establishing this vector, it is included in the data structures associated with a driver.

1.3.1 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 a 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 DCB's 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.

1.3.2 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 done exclusively by either the Executive or driver, not both.

1.3.3 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 a controller per line since all lines may transfer in parallel. Most of the information in the SCB is dynamic. The SCB is used by both the Executive and the driver.

1.3.3.1 Interrelation of the I/O Control Blocks - Without explicit details on the contents of the DCB, UCB, and SCB, their relationships are difficult to correlate. This section is intended to represent their interrelationship without entering into the detailed contents of the control blocks, leaving such a discussion to be pursued in Chapter 3.

Figure 1-2 shows the data structure that would result for three LA30 DECwriters interfaced via DL11 controllers. The structure requires one DCB, three UCB's, and three SCB's, since the activity on all three units can proceed in parallel.

In Figure 1-3, the internal data structure for an RK11 disk controller with three units attached is depicted. Note that only one SCB exists because only one of the three units may be active at any given time.

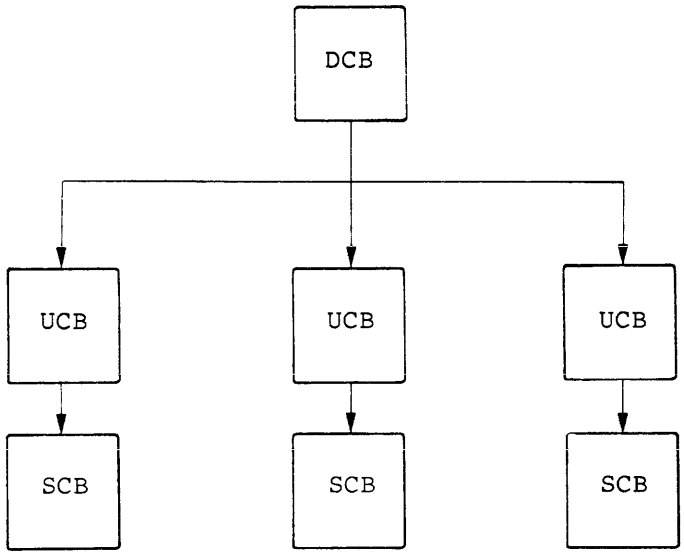

Figure 1-2
DL11 Disk I/O Data Structure

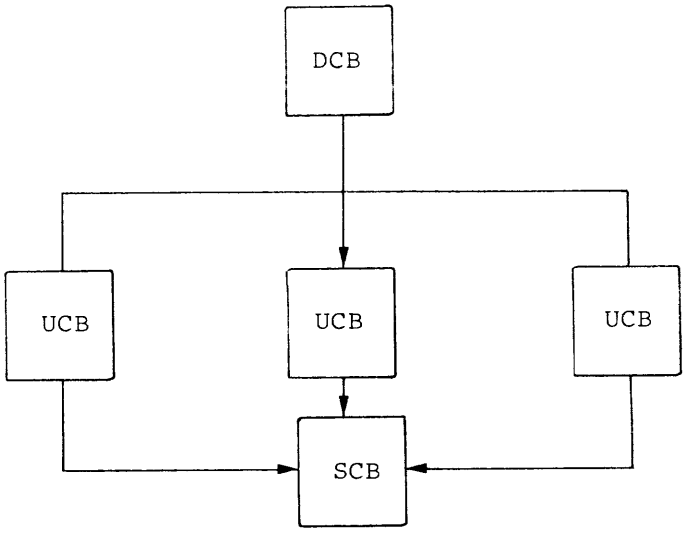

Figure 1-3
RK11 Disk I/O Data Structure

Taken together, Figures 1-2 and 1-3 illustrate the strategy underlying the existence of three basic I/O control blocks. There need be only one DCB per device type. The SCB, depending on the degree of parallelism that is desired or possible, can exist for each device-unit, or only once for controllers servicing multiple device-units.

As will be seen later, this data structure has the effect of providing considerable flexibility in configuring I/O devices, and, due to the information density in the structures themselves, substantially reduces the code requirements of the associated drivers.

1.3.4 The I/O Packet

An I/O Packet contains information extracted from the QIO DPB and other information needed to successfully initiate and terminate I/O requests.

1.3.5 The I/O Queue

Each time an I/O request is made, the Executive is entered, and, if a series of validity checks proves successful, the Executive will generate a data structure called an I/O Packet. The Executive will then insert 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 de-queue the next I/O Packet and deliver it to the requesting driver. The driver never directly manipulates the I/O Queue.

1.3.6 The Fork List

All drivers in RSX-11M can easily be written as multi-controller drivers; all drivers may run in parallel with other drivers and with themselves. When independent processes may execute in parallel, a method for synchronizing their access to common data bases is essential. At the Executive level in RSX-11M, a process may synchronize its access to a data base by requesting the Executive to transform it into a fork process. Such an operation creates a specialized context for the process and places it into a list called the Fork List. Processes in the Fork List are granted FIFO access to common data bases. Once granted access to the data base, the process is guaranteed control of the data base until it relinquishes it by exiting. Not until the process exits will the next process in the Fork List be granted data base access. Thus, it is via the fork mechanism and the associated Fork List that access to shared system data bases is synchronized. Essentially, of the two basic techniques available for data base access synchronization:

1. Interrupt lockout, and

2. Access queuing,

RSX-11M has chosen the latter.

1.3.7 The Device Interrupt Vector

The device interrupt vector is initialized when defining data structures, and not dynamically. This makes the driver code independent of device register address assignments and 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 7. See lines 81 thru 85 in section 5.2.1 (section 5.2.1 contains the source code for the data structure of a sample driver).

1.4 EXECUTIVE SERVICES

The I/O-driver-related services provided by the Executive 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 this processing is to extract from the driver all I/O support functions not directly related to the actual issuance of a function request to a device. If the outcome does not result in the queueing of an I/O Packet to a driver, the driver is unaware that a QIO directive was ever issued. As will be shown shortly, many QIO directives do not result in the initiation of an I/O operation.

1.4.1 Pre-Driver Initiation Processing

QIO directive processing performs the following pre-driver initiation services:

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

2. If the LUN is valid, check if 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, the Executive 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 subject event flag is reset, and the IOSB is cleared.

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 directive status is set to +1.

   Note that directive rejections in any of the above steps are completely transparent to the driver.

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 I/O Finish routine ($IOFIN) is called. $IOFIN sets status and clears the QIO request from the system.

7. If the requested function does not require a call to the driver, appropriate actions are handled by the Executive and $IOFIN is called.

8. If all I/O Packet checks are positive, the I/O Packet is placed in the driver queue according to the priority of the requesting task.

1.4.2 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 I/O drivers. These services are discussed in detail in Chapter 4.

There are, however, four Executive services that merit special emphasis, since they are used by virtually every driver in the system:

1. Interrupt Save ($INTSV);

2. Get Packet ($GTPKT);

3. Create Fork Process ($FORK), and

4. I/O Done ($IODON).

1.4.2.1 Interrupt Save ($INTSV) - Interrupts from a device are fielded directly by the driver. Immediately following the interrupt, the driver is operating at hardware priority level 7 and is, therefore, non-interruptible. If the driver needs a lengthy processing cycle (greater than 100us) to process the interrupt or requires registers, it should call $INTSV; this has the effect of stacking external interrupts, altering the hardware priority, and providing the calling routine with two free registers to use in processing the interrupt. More will be said about $INTSV in section 1.5.

1.4.2.2 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. When work is available, $GTPKT delivers to the driver the highest priority, executable I/O Packet in the driver's I/O queue, and sets the SCB status to busy. If the driver's I/O Queue is empty, $GTPKT returns a no-work indication.

If the SCB related to the device is already busy, $GTPKT so informs the driver, in which case the driver immediately returns control to its caller.

To the driver writer, note that no distinction exists between no-work and SCB busy, since, in either case, an I/O operation cannot be initiated.

1.4.2.3 Create Fork Process ($FORK) - Synchronization of access to shared data bases is accomplished via a fork process. When a driver needs to access a shared data base, it must do so as a fork process; the driver creates a fork process by calling $FORK.

1.4.2.4 I/O Done ($IODON) - At the completion of an I/O request, a number of centralized checks and additional functions are performed:

- 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 also declares a significant event, resets the SCB and device unit status to idle, and releases the dynamic storage used by the completed I/O operation.

1.5 PROGRAMMING STANDARDS

RSX-11M I/O drivers are integral components of the RSX-11M Executive. As such, 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 programmers to incorporate I/O drivers into their systems. Failure to observe the internal conventions and protocol will result in poor service and reductions in system efficiency.

1.5.1 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, multi-unit, multi-controller) and with other Executive processes executing in parallel.

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.

1.5.2 Programming Conventions

The programming conventions used by RSX-11M system components are identical to those described in Appendix E of the RSX-11 MACRO-11 Reference Manual. Users preparing I/O drivers for incorporation into an RSX-11M system are strongly urged to adhere to these conventions.

1.5.3 Programming Protocol

Since 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.

When a device interrupts, an I/O driver is entered directly, usually calling $INTSV for common save and state switching services. (Two states, user and system, exist in RSX-11M; the conventions discussed in this manual generally refer to processes running from the system state, since drivers operate entirely in system state.) At the completion of the services provided by $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 if it is necessary to switch tasks). The Fork Processor linearizes access to shared system data bases. The details of all these routines will be covered in Chapter 4.

The interrupt vectors in lower 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 four 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 data base source must set up the interrupt vector). The driver then starts executing with interrupts locked out. A driver itself may be executing at one of three levels of interrupt sensitivity:

1. At priority 7 with interrupts locked out;

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

3. At fork level which is at priority 0.

1.5.3.1 Processing at Priority 7 with Interrupts Locked Out - By internal convention, processing at this level (interrupt processing routine level 1) is limited to 100us. If processing can be completed in this time, then the driver simply dismisses the interrupt by executing an RTI instruction. The interrupt has been processed and dismissed with minimal overhead.

1.5.3.2 Processing at the Priority of the Interrupting Source - If the driver requires additional processing time or the use of general purpose registers, it calls the routine $INTSV (Interrupt Save). The priority at which the caller is to run immediately follows the call to $INTSV. The driver should set this priority level to that of the interrupting source.

$INTSV save uses the specified priority to set up the interrupt routine such that it is interruptible by priorities higher than that of the interrupting source and conditionally switches to system state if the processor is not already in system state.

The saving of general registers R4 and R5 is done to free these registers for the driver. It is an internal programming convention that assumes the driver will not require more than two registers during interrupt processing. If it does, it must save and restore any additional registers it uses. Processing time following the return from $INTSV should not exceed 500us*.

In actual practice, every driver in the system calls $INTSV on every interrupt after executing perhaps 0 or 1 instructions (such as saving the PS if more than one controller is being driven). This is due to two factors:

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

---

* The 500us period is a guideline. The shorter the period of time a realtime executive spends at an elevated priority level, the less responsive the system will be to devices of lower priority. This 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 may safely be executed at the priority of the interrupting source, which is, of course, desirable.

1.5.3.3 Processing at Fork Level - A driver calls $FORK to meet the requirement to become fully interruptible (so as to conform to the 500us time limit) or to access a shared system data base. $INTSV must be called prior to calling $FORK.

By virtue of calling $FORK, the routine is now at processing level 3 (interruptible) and its access to system data bases is strictly linear. The Fork List is a list of system routines waiting to complete their processing, in particular, waiting to access a shared system data base. At fork level, all registers are available to the process, and R4 and R5 retain the contents they had on entrance to $FORK.

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

1. Registers are not available for use unless $INTSV is called or the driver explicitly performs save and restore operations. If $INTSV is called, the use of any registers, except R4 and R5, requires that these registers be saved and restored.

2. Non-interruptible 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 via a fork process.

1.6 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 the QIO directive is actually issued. The following assumptions apply:

1. 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.

2. The only I/O request in the system will be the sample request under discussion.

3. The example will progress without encountering any errors that would prematurely terminate its data transfer; thus, no error paths will be discussed.

The I/O flow proceeds as described below:

1: [Task issues QIO directive]

All Executive directives are called via EMT 377. The effect of the EMT is to cause the processor to stack the PS and PC and pass control to the Executive's directive processor.

1a. [First level validity checks]

The QIO directive processor validates the LUN and UCB pointer. Invalid data results in directive rejection.

1b. [Redirect algorithm]

Since the UCB may have been dynamically redirected via 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). If valid, the event flag is reset and the I/O status block is cleared.

2. [Obtain storage for and create an I/O Packet]

QIO directive processing now acquires an 18-word block of dynamic storage for use as an I/O Packet. If the 18-words of storage are obtained, the directive is accepted. It inserts data items into the packet which are subsequently used by both the Executive and 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:

a) Control;

b) No-op;

c) File, or

d) Transfer.

Control functions, with the exception of Attach/Detach, are queued to the driver. No-op functions do not result in data transfers and are performed by the Executive without calling the driver.

A file function may require processing by the file system. More typically, the request is a read or write virtual function which is transformed into a read or write logical function without requiring file system intervention. When transformed into a read or write logical, it becomes a transfer function (read and write logical are, by definition, transfer functions).

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

4. [Driver processing]

4a. [Request work]

To obtain work, the driver calls Get Packet ($GTPKT). $GTPKT will either provide work, if it exists, or inform the driver that no work is available, or the SCB is busy; if no work exists, the driver returns to its caller. If work is available, $GTPKT will set the device controller and unit busy, dequeue an I/O request packet, and return 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 it according to the programming protocol described earlier. 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 (4b). When the processing of an I/O request is complete, $IODON is called.

6. [I/O Done processing]

$IODON removes the device unit and controller busy status, 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 looking 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 process which will issue a QIO directive, starting the I/O flow anew.

1.7 DATA STRUCTURES AND THEIR INTERRELATIONSHIPS

This section presents all the individual control blocks, 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. $DEVTB word, the Device Control Block (DCB), and the Driver Dispatch Table (DDT);
5. Unit Control Block (UCB);
6. Status Control Block (SCB), and
7. Volume Control Block (VCB).

Figure 1-4, which will provide 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 guide the reader through the data structures.

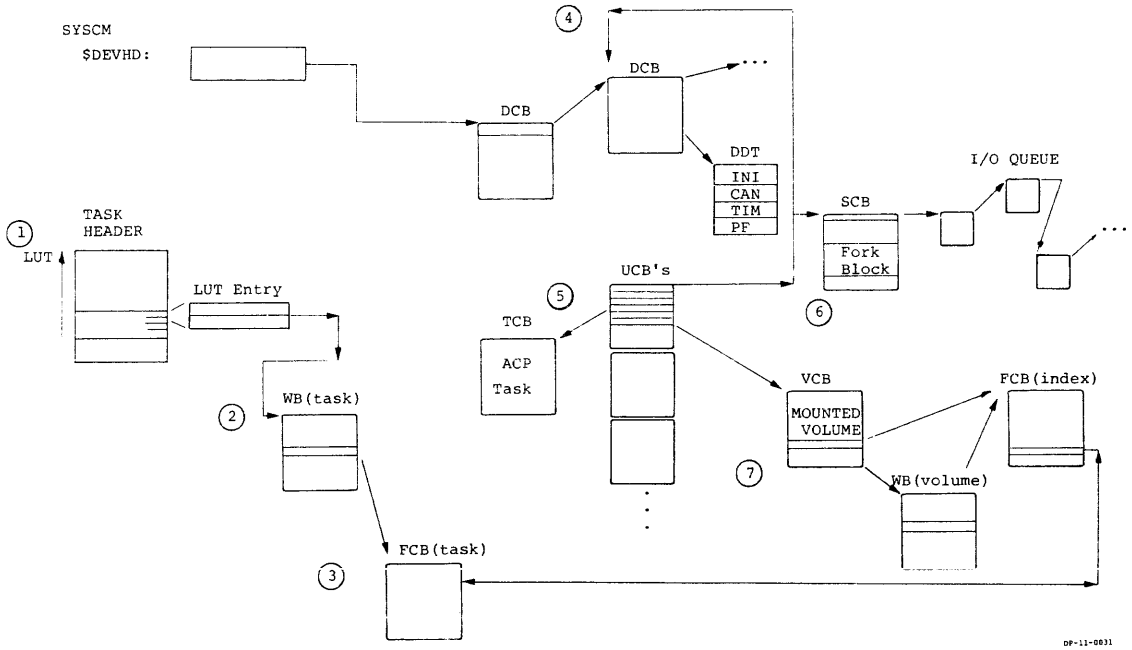

Figure 1-4
I/O Data Structure

1. The Task Header, one of two independent entries in the I/O data structure, is constructed during the task-build process. The 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 and places 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.

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 where the file resides.

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. $DEVTB is a word located in system common (SYSCM) and is the other independent entry in the I/O data structure. $DEVTB points to a singly-linked, uni-directional list of Device Control Blocks (DCB's). Each device type in a system has an associated DCB. At least one DCB exists per device type. The DCB list is terminated by a zero in the link word.

   Linked to each DCB is a Driver Dispatch Table (DDT). The DDT contains the addresses of the driver's four callable entry points.

5. A key data structure is the Unit Control Block (UCB). All the UCB's associated with a DCB appear in consecutive memory locations. During internal processing of an I/O request, R5 contains the address of the related UCB, and it is via 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. A driver services a fixed set of UCB's. When servicing a request for one of its UCB's, it is unaware of whether I/O was issued directly to the UCB or whether it was issued to a UCB 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.

7. One Volume Control Block (VCB) exists for each MOUnted volume in a system. The VCB is used to maintain volume-dependent control information. It contains pointers to the File Control Block (FCB) and Window Block (WB) used to control access to the volume's index file. (The index file is a file of file headers.) The WB for the index file serves the same function as the WB for a user file. All unique FCB's 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 which 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 his application requires.

1.7.1 Data Structures Summary

The writer of a conventional driver never manipulates the entire I/O data structure. In fact, he is almost exclusively involved only with the I/O Packet, the UCB, and the SCB. The entire structure has been presented to add depth to the driver writer's understanding of the I/O system, to emphasize the relationships among individual control blocks, and to further clarify the role a given driver fulfills in the processing of an I/O request.

CHAPTER 2

INCORPORATING USER-WRITTEN DRIVERS INTO RSX-11M

2.1 INTRODUCTION

Though explicit details for writing a driver have not been presented, enough conceptual information now exists to consider incorporating a user driver into a system. This follows from the fact that many considerations for writing a driver are most easily presented within the context of the process followed for installing it.

The reader is already assumed to be familiar with the RSX-11M System Generation Manual.

Taken together, Chapters 1 and 2 present a comprehensive overview of the relevant considerations for undertaking the implementation of a driver.

2.2 OVERVIEW

The user who has decided to add a driver to RSX-11M has concomitantly shared the responsibilities for the integrity of the Executive. Errors in this code can easily cause a system failure and bring to a halt all user service.

The basic steps involved in creating and installing a user-written driver are 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 and its associated data structures.
4. Run Sysgen Phase 2.

At the completion of Sysgen Phase 2, the user has a system with the user-written driver integrated into it. Since it is anticipated that a debugging sequence will be required to shake down the driver, the following sequence will result in an updated driver being incorporated into the existing system:

1. Correct and re-assemble the driver and/or data structures.
2. Run the Librarian to replace the old object modules in RSX11M.OLB with the repaired ones.
3. Rebuild the Executive using RSXBLD.CMD.
4. Using Virtual MCR, rebuild the system.
5. Bootstrap the system.

When adding a user-written driver to the system, the driver may be assembled to include padding space for possible expansion during the debugging process.

2.3 INCORPORATING A DRIVER - DETAILS

2.3.1 Creating the Data Structure

The data structures associated with I/O drivers will be detailed in Chapter 3. Of the structures discussed, only three require assembly source:

1. The DCB;
2. UCB's, and
3. SCB's.

Within these control blocks, only those items which are static or require initialization are supplied in the source description. Listed below is an overview of the data fields the driver writer will be required to supply in the assembly source of his driver's data structure.

2.3.1.1 Required Device Control Block (DCB) Fields - The required DCB fields are described below:

D.LNK — Link to next DCB.

This field will be zero if this is the last DCB. If the user is incorporating more than one user-written driver at one time, then this field should point to another DCB in the DCB chain.

D.UCB — Address of the first UCB associated with this DCB.

D.NAM — Two-character ASCII generic device name. This name must be unique.

D.UNIT Highest and lowest unit numbers controlled by this DCB.

D.UCBL Length of a UCB.

If a given DCB has multiple UCB's, all UCB's 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 will contain a global reference to the label associated with this table.

D.MSK I/O function masks

The user must supply bit-by-bit mapping for these four I/O function masks. Note that the format of the mask words is split into two groups of four words. Functions 0-15 are covered by the first group, and functions 16-31 by the second.

2.3.1.2 Required Unit Control Block (UCB) Fields - The required UCB fields are described below:

U.DCB Backpointer to the associated DCB.

U.RED Initially contains the address of this UCB (i.e., redirect pointer).

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 (Chapter 3) 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.

2.3.1.3 Required Status Control Block (SCB) Fields - The required SCB fields are described below:

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 (i.e., controller number multiplied by 2).
S.STS	Controller status.
S.CSR	Address of control and status register.

2.3.1.4 Source Format of the Data Structure - A single DCB can service multiple UCB's and SCB's. The existence of multiple UCB's and SCB's is determined by the actual device subsystem being supported by a given driver on the user's operational hardware configuration. Figures 1-2 and 1-3 illustrate possible DCB, UCB, and SCB structural relationships. Typically, in writing a data structure source (DEC-supplied RSX-11M drivers use this scheme), the DCB is placed first, followed by the UCB(s), followed by the SCB(s).

2.3.2 Creating the Driver Source Code

Creating the source code to drive a device involves the following:

1. Thorough reading and understanding of this manual;
2. Detailed familiarization with the physical device and its operational characteristics;
3. Determining the level of support required for the device;
4. Creating the data structure source code for the device;
5. Determining actions to be taken at the five driver entry points:
    a. Initiator;
    b. Cancel I/O;
    c. Timeout;
    d. Powerfail, and
    e. Interrupt.
6. Creating the driver source code.

Source code for the driver and data structure should be created on the object disk under UIC [200,200].

2.3.3 Incorporating the User-Written Driver

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

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

if answered affirmatively, results in a second query

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

The answer to which is required so Phase 1 can allocate sufficient vector space to avoid run-time destruction of the system stack.

At the completion of Phase 1, Phase 2 is entered. During the execution of 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 the driver writer through the process of adding his 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, viz., [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
>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:
>;
>;       [1,2x]RSX11M/LB:XXDRV
>;
>; INTO THE COMMAND FILE IN THE PLACE WHERE THE
>; OTHER DRIVERS ARE REFERENCED.  XXDRV REPRESENTS THE NAME OF
>; YOUR DRIVER(S).  IN ADDITION, LOCATE THE LINE IN WHICH THE
>; MODULE SYSTB IS REFERENCED AND ADD A REFERENCE TO YOUR
>; USRTB MODULE IMMEDIATELY AFTER IT.  E.G.:
>;
>;       [1,2x]RSX11M/LB:LOADR:NULTK:SYSTB:USRTB:SYTAB:INITL
>;
>; THEN 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

*↵
[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]
```

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

2.4 DRIVER DEBUGGING

Since the protection checks afforded user programs are not available to system modules, driver errors will be more difficult to isolate than user program errors. But conventional drivers, being modular and short, should be easily debugged. The following three sections describe a set of debugging tools and guidelines that should simplify the driver debugging process.

Section 2.4.1 describes how to re-incorporate a driver into a system after a fault has been discovered. Section 2.4.2 describes the Executive Debugging Tool, and Section 2.4.3 provide some general hints for isolating faults in Executive software (of which drivers are a subset).

2.4.1 Rebuilding and Re-incorporating the User Driver

Rebuilding and re-incorporation involves five steps:

1. Correction and re-assembly of the driver and/or device data structures;
2. Updating the Executive object module library;
3. Rebuilding the Executive;
4. Running Virtual MCR to rebuild the system, and
5. Bootstrapping the new system.

2.4.1.1 Re-Assembly - 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:

```
>RUN $MAC/UIC=[200,200]
MAC>XXDRV=[1,1]EXEMC/ML,[200,200]RSXMC,XXDRV
MAC>USRTB=[1,1]EXEMC/ML,[200,200]RSXMC,USRTB
MAC>^Z
```

will effect the re-assembly of both the driver and data base.

2.4.1.2 Updating the Executive Object Module Library - After re-assembly of the user driver and/or data base, the Executive object module library must be updated. The following commands will accomplish this:

```
>RUN $LBR/UIC=[1,2x]*
LBR>RSX11M/RP=[200,200]XXDRV,USRTB
LBR>^Z
```

---

* 'x' is a '0' for an unmapped system and '4' for a mapped system.

2.4.1.3 Rebuilding the Executive - Since an updated driver is to be inserted, the Executive, of which the driver is a part, must be relinked. To do this, enter the following commands:

```
>RUN $TKB/UIC=[1,2x]*
TKB>@RSXBLD
TKB>^Z

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

2.4.1.4 Incorporating Tasks into the System - Run Virtual MCR (VMR) using the dialog shown as a guide. On completion, the new system is ready for bootstrapping. The general procedure to follow is:

1. Establish system partitions;
2. Release all unused space to the dynamic storage region;
3. Install tasks (at least FCP, INS, MOU, and MCR); and
4. Exit from Virtual MCR and boot in the new system.

VMR Example:

```
>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 FCP                ! 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
```

2.4.1.5 Bootstrapping the New System - The new system may now be bootstrapped with the MCR Boot command, as shown below:

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

---

* 'x' is a '0' for an unmapped system and '4' for a mapped system.

2.4.2 RSX-11M Executive Debugging Tool

An interactive debugging tool has been developed for RSX-11M to aid in the debugging of executive modules, I/O drivers, and interrupt service routines. This debugging aid is called XDT and is a version of RSX-11 ODT, which does not contain the following features and commands:

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 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

The X (Exit) and P (Proceed) commands have also been changed. The X command causes a jump to the crash reporting routine, and the P command will permit the user to proceed if an unknown breakpoint is encountered.

Other than the omitted features and the change in the definition of the X and P commands, XDT is command-compatible with RSX-11 ODT, and the RSX-11 ODT Manual may be used as a guide to its 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:

XDT: <system version>

followed by the prompting symbol

XDT>

on the console terminal. Breakpoints may be set at this time, and then a G command may be given, returning control to the RSX-11M Executive initialization code. Whenever a breakpoint is reached, a printout similar to that of RSX-11 ODT will occur.

A forced entry to XDT can be executed at any time from a privileged terminal via 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:

```
BE:xxxxxx
```

followed by the prompting symbol

```
XDT>
```

on the console terminal. Breakpoints and/or other XDT commands may then be executed. System operation may be continued 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 is done via the console terminal, and the L (List Memory) command can list on either the console or the line printer.

2.4.3 Fault Isolation - Some General Hints

2.4.3.1 Introduction - 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, these bugs are often difficult to diagnose. It is extremely important that the driver writer develop the skills and discipline needed to rapidly isolate the source of a system failure.

2.4.3.2 Fault Classifications - Four culprits can be identified when the system faults:

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

The immediate action on the part of the driver-writer subject to one of these errors is to determine which of these four cases is the source of the fault. Of prime concern will be the procedures which may help the driver writer uncover the fault source. The repair of the fault itself is assumed to be the driver writer's responsibility.

Faults manifest themselves in roughly three ways (they are listed here in order of increasing difficulty of isolation):

1. The system displays the CRASH printout and halts or, if XDT has been included, an unintended trap to XDT occurs.

2. The system halts but displays nothing.

3. The system is in an unintended loop.

2.4.3.3 Immediate Servicing - RSX-11M can be built to contain resident crash reporting and panic dump routines; the following discussions assume the existence of such a system. (Note that the minimal system will not have space for these routines). Section 2.5 contains sample listings from both crash and panic dump routines.

The immediate aim, regardless of the fault manifestation, is to initiate the crash reporting and panic dump routines.

CASE 1 - The system has trapped to XDT:

The trap may or may not be intended (e.g., a previously set breakpoint). If it is not intended, type the X command, causing XDT to exit to the crash reporting routine; if, however, the source of the problem is suspected (for example, a recent coding change), then pertinent data structures and code may be examined using XDT.

CASE 2 - The System Has Displayed the Crash Printout:

In this case, all the basic information describing the state of the system has been displayed. The actual Crash printout will be described after learning how to invoke Panic Dump for cases 3 and 4 (see below).

CASE 3 - The System Has Halted - No Information Displayed:

Before taking any action, preserve the current PS and PC and the pertinent device registers (i.e., 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 obtaining the PS and PC, invoke the Panic Dump Routine by entering 40(8) in the switch register, depressing LOAD ADDR, and then START.

The value 40(8) is the address of a JMP to the Panic Dump Routine in all RSX-11M systems.

The Panic Dump Routine saves registers R0 through R6 and then halts, awaiting dump limits to be entered via the console switch register. The PS is cleared when START is depressed, and the original PC is destroyed; thus, the importance of recording these vital pieces of debugging information before initiating the Panic Dump Routine should be recognized.

Dumps of selected blocks of memory may be obtained using the following procedure:

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

2. Enter the high dump limit in the console switch register and depress 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.

   To reach the same status arrived at with crash reporting in Case 2 above, enter the dump limits 0-520(8) when the panic dump routine first halts. This will dump 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 may be obtained from the module LOWCR in the Executve memory allocation map.

CASE 4 - System Is in an Unintended Loop:

Proceed as follows:

1. Halt the processor

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

After recording the PS and PC, the driver-writer may want to step through a number of instructions in an attempt to locate the loop.

After the attempt to locate the loop, transfer to the panic dump routine as in Case 3.

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

2.4.3.4 Other Pertinent Fault Isolation Data - Before proceeding with the task of locating the fault, the driver-writer is strongly advised to dump system common (SYSCM). This can be accomplished by looking for the module SYSCM in the Executive memory allocation map and entering the appropriate limits to the Panic Dump Routine. SYSCM contains a number of critical pointers and listheads.

In addition, the driver-writer should dump the dynamic storage region and the device tables. The dynamic storage region is the module INITL and the device tables are in SYSTB.

The driver-writer now has:

PS

PC

The Stack

R0 through R6

Pertinent device registers

The dynamic storage region

The device tables, and

System common.

This data is a minimal requirement for effective fault isolation.

2.4.3.5 Fault Tracing - Three pointers in SYSCM are critical in fault tracing. These pointers are described below:

$STKDP - Stack Depth Indicator

This data item will indicate which stack was being used at the time of the crash. As will be seen, this 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 managed to crash the system. In a system with brickwall protection (for example, the mapped RSX-11M system), the non-privileged 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.

Locating the task header provides additional data. The first word in the header is the user's stack pointer the last time it was saved. If the user branches wildly into the Executive, it will terminate the user task, but the system will continue to function (possibly erroneously). Knowing the user's stack pointer provides one more link in the chain which 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, which is the last word in the header and is pointed to by H.GARD.

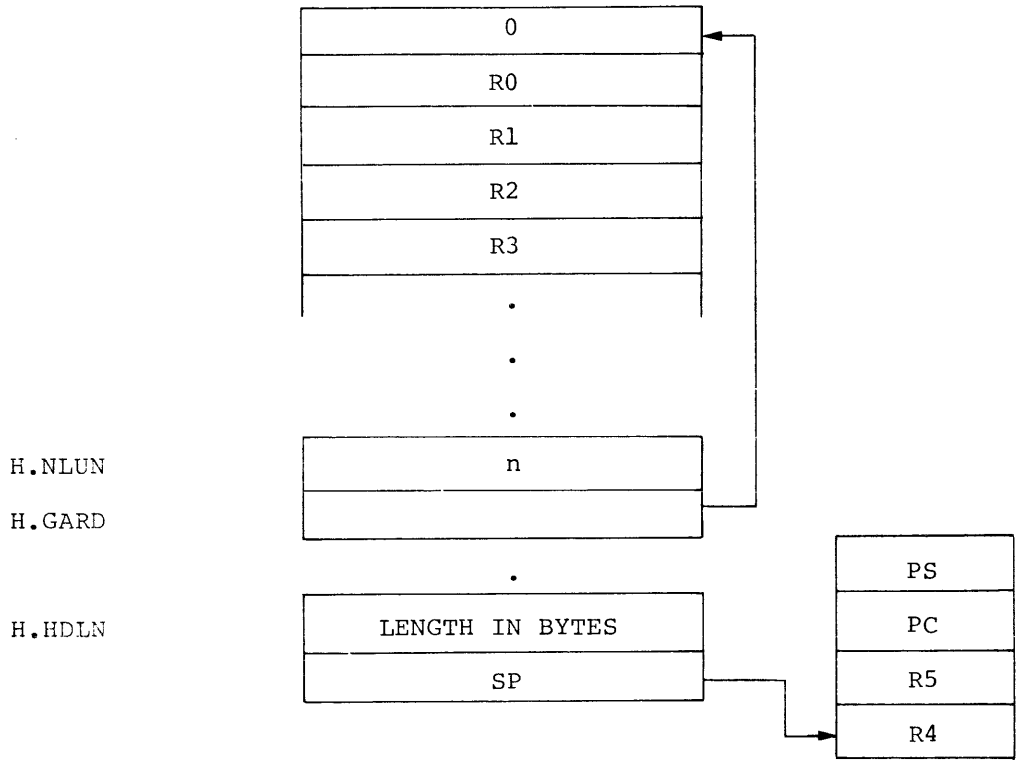

Figure 2-1
Unmapped System Header

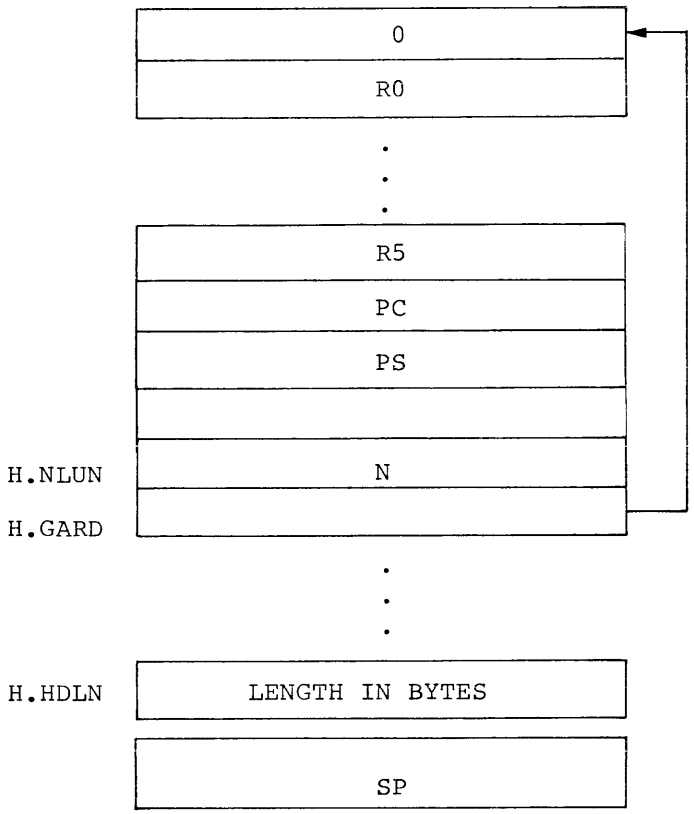

Figure 2-2
Mapped System Header

A. Tracking Faults Following Automatic Display of System State (Cases 1 and 2):

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 will be 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 was issued at a stack depth of zero or less, or a trap instruction was executed at a depth of less than zero. The stack structure in the case of an internal SST fault is shown in Figure 2-3.

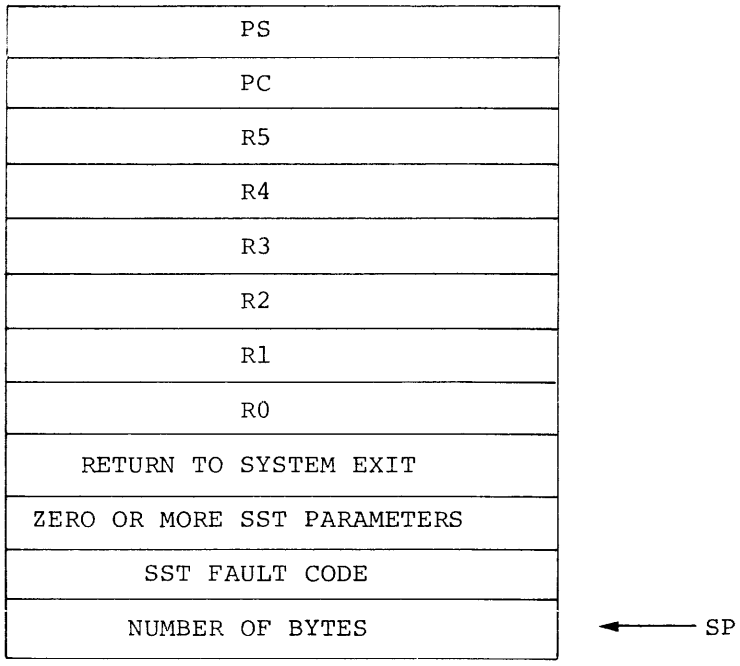

Figure 2-3
Stack Structure - Internal SST Fault

The fault codes are:

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

The PC points to the instruction following the one which caused the SST failure. The number of bytes is the number of bytes that are normally transferred to the user stack when the particular type of SST occurs. If the number is 4, then just the PS and PC are transferred and there are no SST parameters.

If the failure is detected in $DRDSP, the stack is the same as Figure 2-3, except the number of bytes, SST fault code (the fault codes are listed above), and SST parameters are not present. The crash report message, however, will indicate that the failure occurred in $DRDSP.

There is one SST-type failure, stack underflow, that will not have the stack structure of Figure 2-3. To determine where the crash occurred, first establish the stack structure; this can be deduced by the value of the stack pointer (SP) and the contents of the top word on the stack. If the stack structure is that of Figure 2-3, then the failure occurred in $DRDSP, or was a normal SST crash. If the stack structure is determined to be that of Figure 2-4, then a non-normal SST crash has occurred.

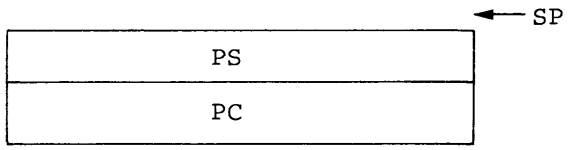

Figure 2-4
Stack Structure - Non-Normal SST Fault

Non-normal SST failures occur when it is not possible to push information on the stack without forcing another SST fault. When this 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 stack pointer will appear to be 4 greater than it really is (i.e., as in Figure 2-4).

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

B. Tracking Faults When the Processor Halts Without Providing Fault Display:

Tracking starts with an examination of $STKDP, $TKTCB, and $HEADR. The difficulty in tracking failures in this case is that the system stack is not directly associated with the cause of a failure.

By examining $STKDP, one 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 process focuses on scanning the stack for addresses which may turn out to be subroutine links which will ultimately lead to a thread of events isolating the fault. This is essentially the same aim in looking at the system stack if $STKDP is zero or less.

Frequently, a fault will occur such that the SP points to Top of Stack (TOS)+4. This results from issuing an RTI when the top two items on the stack are data; this will result in a wild branch, then, most probably, a halt. Figure 2-5 shows a case, where two data items are on the stack when the programmer 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 previous SP.

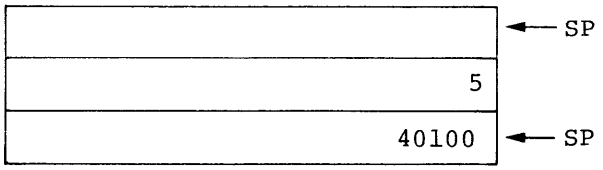

Figure 2-5
Stack Structure - Data Items on Stack

This type of fault also occurs when an RTS instruction is executed with an inconsistent stack. However, in this case SP will point 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.

C. Tracking Faults When an Un-Intended Loop Has Occurred:

After halting the processor, the same state exists as in the preceding section. Some specific suggestions are to check for a stack overflow loop. Patterns of data successively duplicated on the stack indicate a stack looping failure.

D. Additional Hints:

Also of value 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) will define 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. Since all I/O packets are built in system dynamic memory, when they are successfully terminated, their memory is returned to the dynamic memory region. Following the link pointers in this region may reveal whether or not I/O completion proceeded to that point. 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 ultimately produces a double de-allocation of the same packet memory. Double de-allocation of a dynamic buffer in RSX-11M causes a loop in the module $DEACB on the next de-allocation (of a block of higher address) after the second de-allocation of the same block. At that time, R2 and R3 both contain the address of the I/O Packet memory which has been doubly de-allocated.

2.5 SAMPLE OUTPUT FROM CRASH AND PANIC DUMP ROUTINES

2.5.1 Crash Output

A sample of Crash output is shown below:

```
SYSTEM CRASH AT LOCATION 047622

REGISTERS

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

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

SYSTEM STACK DUMP

 LOCATION CONTENTS

 000472   000004
 000474   000000
 000476   001514
 000500   000340
 000502   177753
 000504   000353
 000506   000000
 000510   000000
 000512   057750
 000514   002504
 000516   030011
 000520   100340
 000522   000340
 000524   056446
 000526   000000
 000530   102144
 000532   000000
 000534   101376
 000536   101372
 000540   102022
 000542   170000
```

2.5.2 Panic Dump Output

A portion of the output from Panic Dump is shown below. Output is in three 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 is the octal word value; the second line is the two octal byte values of the word; the third line contains the ASCII representation of the bytes. The ASCII representations are reversed to improve readability. The first output grouping from Panic Dump displays, 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       ^@
```

CHAPTER 3

WRITING AN I/O DRIVER - PROGRAMMING SPECIFICS

In Chapter 1, overviews were given for:

Data structures;

Executive services, and

Programming protocol.

In this chapter, the details for the data structures and Executive services are given. The protocol coverage in Chapter 1 was, however, detailed enough to make further elaboration of programming protocol unnecessary.

3.1 DATA STRUCTURES

Of all the control blocks in the I/O data structure, only four are of direct concern to a driver writer:

1. The I/O Packet;
2. The DCB;
3. The UCB, and
4. The SCB.

Although the data structures contain an abundance of data pertaining to input/output operations, drivers per se are involved only with a subset of this data. Most of the data which requires the driver writer's attention is supplied in the data structure source code, and is not referenced during driver execution. Such an item is classified as:

<initialized, not referenced>*

* The first field states whether the field is initialized in the data structure source, and the second field gives the typical access at execution time.

Fields supplied statically in the source code at the creation of the data structure, and subsequently referenced during driver execution, are classified:

<initialized, read-only>.

Fields set up during driver execution are classified:

<not initialized, read-only>

This form implies that either an agent other than the driver has established the field or that the driver has set it up once and references it read-only thereafter.

or:

<not initialized, read-write>.

Fields which do not involve the driver writer at any level are classified

<not initialized, not-referenced>.

These classifications cover most-likely cases, since exceptions do exist and are appropriately noted.

3.1.1 The I/O Packet

Figure 3-1 is a layout of the 18-word I/O Packet which is constructed and placed in the driver I/O queue by QIO directive processing and subsequently delivered to the driver by a call to $GTPKT. Figure 3-2 is the DPB from which the I/O Packet is generated.

---

* The first field states whether the field is initialized in the data structure source, and the second field gives the typical access at execution time.

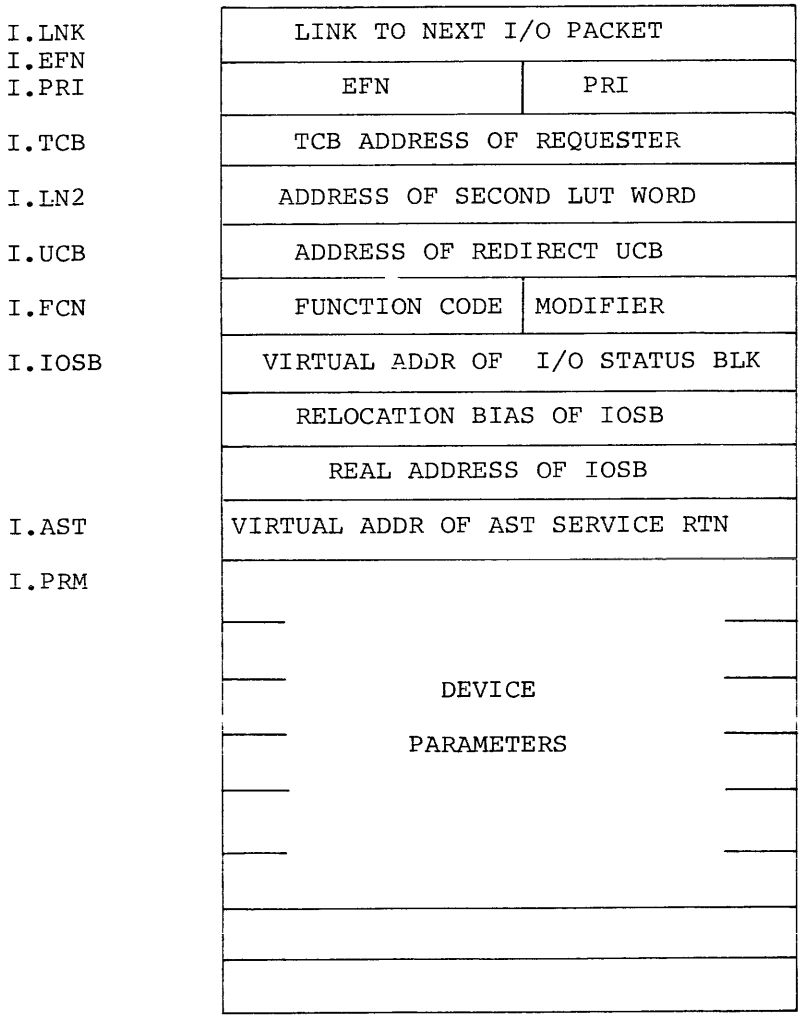

Figure 3-1
I/O Packet Format

3.1.1.1 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 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.PRI

Driver access:

Not referenced.

Description:

Priority copied from the TCB of the requesting task.

I.EFN

Driver access:

Not referenced.

Description:

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

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 was 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 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 3-1) for the I/O request.

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 not.

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 32-word block number 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 where the IOSB started 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 Page Address Register 6. Again, see Appendix A for details.

The deferral of a discussion of the address doubleword to an appendix reflects the fact that a writer of a conventional driver has almost no need to concern himself with the contents or format of the address doubleword. 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 copied from the DPB.

The QIO Directive Parameter Block (DPB) is constructed as shown in Figure 3-2.

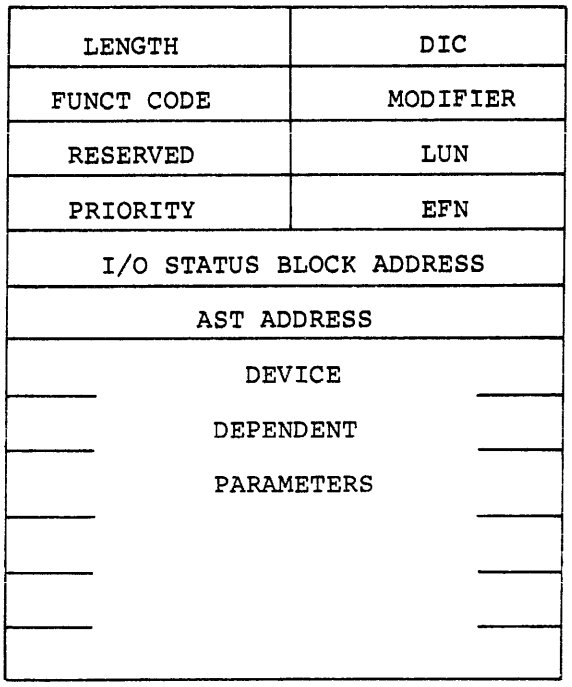

Figure 3-2
QIO Directive Parameter Block (DPB)

The parameters in the DPB have the following interpretation.

Length (required):

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

DIC (required):

Directive Identification Code. For the QIO directive, this value is a 1.

Function Code (required):

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

Modifier:

Device dependent modifier bits.

Reserved:

Reserved byte and must not be used.

LUN (required):

Logical Unit Number.

Priority:

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

EFN (optional):

Event flag number.

I/O Status Block Address (optional):

This word contains a pointer to the I/O status block, which is a 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 zero, 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, these are:

Buffer address

Byte count

Carriage control type

Logical block number

Any optional parameters that are not specified should be filled with zeros.

3.1.2 The Device Control Block (DCB)

Figure 3-3 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.

D.LNK	LINK TO NEXT DCB (Ø=LAST)	
D.UCB	LINK TO FIRST UCB	
D.NAM	GENERIC DEVICE NAME	
D.UNIT	HIGHEST UNIT #	LOWEST UNIT #
D.UCBL	LENGTH OF UCB	
D.DSP	ADDR OF DRIVER DISPATCH TABLE	
D.MSK	LEGAL FCN MSK BITS Ø-15.	
	CONTROL FCN MSK BITS Ø-15.	
	NO-OP'ED FCN MSK BITS Ø-15.	
	ACP FCN MSK BITS Ø-15.	
	LEGAL FCN MSK BITS 16.-32.	
	CONTROL FCN MSK BITS 16.-32.	
	NO-OP'ED FCN MSK BITS 16.-32.	
	ACP FCN MSK BITS 16.-32.	

Figure 3-3
Device Control Block

3.1.2.1 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 DCB in the chain. The driver writer links his DCB into the system DCB's via the global label $USRTB on his first DCB.

D.UCB (Pointer to First UCB)

Driver access:

Initialized, not referenced.

Description:

Address link to the first and, possibly, the only UCB associated with the DCB. All UCB's, for a given DCB, are in contiguous memory locations and must all be 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.

---

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

Description:

The UCB may 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.

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. Thus, null addresses are not permitted. If the driver does not process a given function, then it simply returns. The driver writer must provide a driver dispatch table in the driver source. The label on this table is of the form $nnTBL and must be a global label. The designation nn 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, and

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
R1 = Address of the I/O Packet

If UC.QUE=0
R5 = UCB address

Interrupts are allowed.

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 are locked out.

At Device Timeout:

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

Device interrupts are locked out.

At Power Failure:

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

Interrupts are allowed.

D.MSK (Function Masks)

Driver access:

Initialized, not referenced.

Description:

There are eight words, beginning at D.MSK, which are of critical importance 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 description which follows applies only to non-file-structured devices, since directions for writing drivers for file-structured devices (drivers which interface to FCP) are not included in this manual. Four masks, 2-words per mask, are described by the bit configurations established by the driver writer for these words.

1. Legal function mask;

2. Control function mask;

3. No-op'ed function mask, and

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 function value in the I/O request is filtered by the Executive through the four mask words. I/O function codes range from 0-31. If the function corresponds to a true condition in a mask word, a bit is set in the mask in the position which numerically corresponds to the function code. Thus, if the function 5 is legal, then bit 5 in the Legal Function Mask is set.

The masks are laid out in memory in two 4-word groups. Each 4-word group covers 16 function codes. The first four words cover the function codes 0-15; the second four words cover codes 16-31. Below is the exact layout used for the driver example in Chapter 5.

```
.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.
```

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 by returning 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 in the class <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 neither control nor no-op, then it is either an ACP function or a transfer function. If a function code may require intervention of an Ancillary Control Processor (ACP), the corresponding bit in the ACP function mask must be set.

In the specific case of read/write virtual functions, the corresponding mask bits may be set at the driver writer's option. If the corresponding mask bits for a read/write virtual function are set, QIO directive processing will recognize that a file-oriented function is being requested to a non-file-structured device and convert 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 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 (i.e., being within the address space of the requesting task) and proper alignment (i.e., word or byte). Also, the number of bytes being transferred is checked for proper modulus (i.e., nonzero and a proper multiple).

Mask Word Creation

The creation of the function mask words involves three steps:

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

2. Check the standard RSX-11M function code values in Table 3-1 for equivalencies. Only function code 0 is mandatory. Function codes 3 and 4, if used, must have the RSX-11M system interpretation. It is suggested that functions having an RSX-11M system counterpart use the RSX-11M code, but this is not required, except in the case where the device is to be used in conjunction with an ACP. From the supported function list, the two legal function mask words can be built.

3. Given the legal function mask,

3a. The Control Function mask is built 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 it either 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. (Buffer addresses in standard format are automatically converted to Address Doubleword format.)

Control functions are, essentially, those whose DPB's do not contain buffer addresses or counts.

3b. The No-op Function Mask is created by deciding which legal functions are to be no-op'ed. Typically, for File Control Services (FCS) compatibility 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.

3c. Finally, the ACP functions Write Virtual Block and Read Virtual Block may be included. Other ACP functions that might be included fall into the non-conventional driver classification and are beyond the scope of this document.

3.1.2.2 I/O Function Codes - The filtering process which cascades through the function mask words in the DCB uses the function code byte supplied in the QIO directive DPB as the match value. Table 3-1 contains the function code values used for DEC-supplied drivers.

Table 3-1
Standard I/O Function Codes

FUNCTION VALUE(8)	EQUATED SYMBOLIC	I/O FUNCTION
0	IO.KIL	Cancel I/O
1	IO.WLB	Write Logical Block
2	IO.RLB	Read Logical Block
3	IO.ATT	Attach Device
4	IO.DET	Detach Device
5		Unused
6		Unused
7		Unused
10		Unused
11	IO.FNA	Find File in Directory
12		Unused
13	IO.RNA	Remove File from Directory
14	IO.ENA	Enter File in Directory
15	IO.ACR	Access File for Read
16	IO.ACW	Access File for Read/Write
17	IO.ACE	Access File for Read/Write/Extend
20	IO.DAC	Deaccess File
21	IO.RVB	Read Virtual Block
22	IO.WVB	Write Virtual Block
23	IO.EXT	Extend File
24	IO.CRE	Create File
25	IO.DEL	Mark File for Delete
26	IO.RAT	Read File Attributes
27	IO.WAT	Write File Attributes
30		Unused
31		Unused
32		Unused
33		Unused
34		Unused
35		Unused
36		Unused
37		Unused

Of the function code values listed in Table 3-1, 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. If QIO directive processing encounters a function code of 3 or 4 and the code is not no-op'ed, it will assume that they represent Attach device and Detach device, respectively. The other codes are suggested but not mandatory. The driver writer is free to establish all other function code values on non-file-structured devices. The mask words must obviously reflect the proper filtering process.

If a driver is being written for a file-structured device, the standard function codes of Table 3-1 must be used.

3.1.3 The Status Control Block (SCB)

Figure 3-4 is a layout of the 13-word SCB. The SCB describes the status of a control unit which can run in parallel with all other control units.

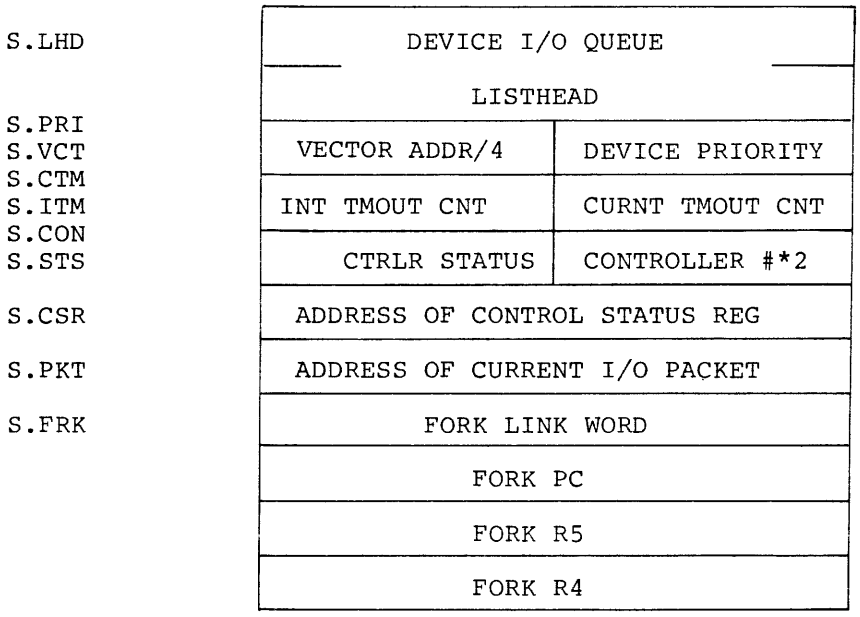

Figure 3-4
Status Control Block

3.1.3.1 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 which form the I/O queue listhead. The first word points to the first I/O Packet in the queue, and the second word points to the last I/O Packet in the queue. If the queue is empty, the first word is zero, and the second word points to the first word.

S.PRI (device priority)

Driver access:

Initialized, not referenced.

Description:

Contains the priority at which the device interrupts.

S.VCT (interrupt vector divided by four)

Driver access:

Initialized, not referenced.

Description:

Interrupt vector address divided by four.

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 will examine active times, decrement them and, if they reach 0, call the driver at its device timeout entry point.

The internal clock count is kept in 1-second increments. Thus, a time count of 1 is not precise, since the internal clocking mechanism is operating asynchronously with driver execution. The only meaningful minimum clock interval is 2 if the programmer intends to treat timeout as a consistently detectable error condition. Note, if the count is 0, that no timeout will occur; it is, in fact, an indication that timeout is not operative. The maximum count is 255. The driver writer is responsible for setting this field. Resetting is 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. Used by drivers which are written to support more than one controller. S.CON may be used by the driver to index into a controller table created and maintained internally to the driver itself. Indexing the controller table enables the driver to service the correct controller when a device interrupts.

S.STS (initialize to zero)

Driver access:

Initialized, not referenced.

Description:

Establishes the controller as busy/not busy. This byte is the interlock mechanism for marking a driver as busy for a specific controller. 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. S.CSR is used by the driver to initiate I/O operations and to access, via 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 this to set the off-line bit at bootstrap so system software can be interchanged between systems without an intervening system generation. Otherwise, it is only accessed by the driver itself.

S.PKT (Reserve one word of storage)

Driver access:

Not initialized, read-only.

Description:

Address of the current I/O Packet established by $GTPKT. This field is used to retrieve the I/O Packet address upon the completion of an I/O request.

S.FRK (reserve four words of storage)

Driver access:

Not initialized, not referenced.

Description:

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

3.1.4 The Unit Control Block (UCB)

Figure 3-5 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.

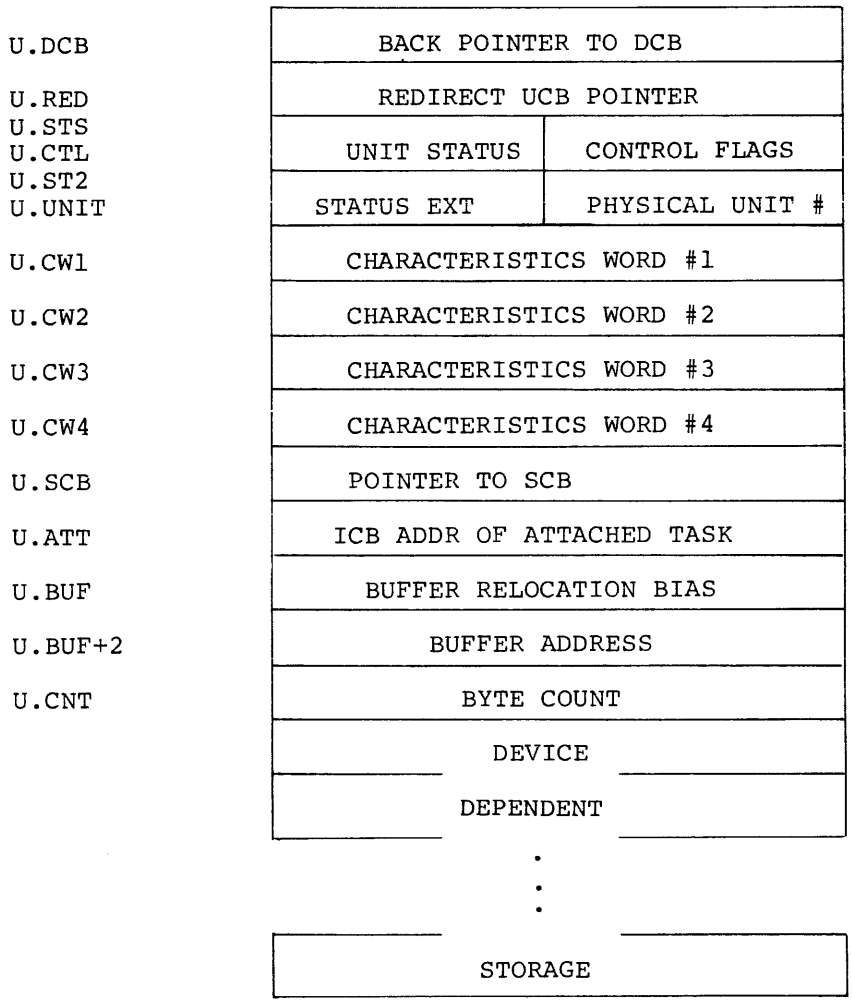

Figure 3-5
Unit Control Block

3.1.4.1 UCB Details - The fields in the UCB are described below:

U.DCB (pointer to associated DCB)

Driver access:

Initialized, not referenced.

Description:

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

U.RED (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 UCB itself.

U.CTL (set by driver writer)

Driver access:

Initialized, not referenced.

Description:

U.CTL and the function mask words in the DCB drive QIO directive processing. The driver writer is totally responsible for setting up this bit pattern. Any inaccuracy in the bit setting of U.CTL will produce erroneous I/O processing. Bit symbols and their meaning 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 will be called when an Attach/Detach I/O function is processed by $GTPKT. Typically, the driver has no 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 to be 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 is to call the driver prior to queueing the I/O Packet. Once gaining to-be-queued control, the disposition of the I/O Packet is the driver's responsibility. Typically, an I/O Packet is queued prior to a call to the driver, which later retrieves it by a call to $GTPKT.

UC.PWF - Unconditional call on power failure bit.

If set, the driver is always to be called when 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.

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 under the discussion of U.BUF below).

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

These two bits are used to check if the byte count specified in an I/O request is a legal buffer modulus.

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

UC.ALG and UC.LGH are independent settings.

UC.ATT, UC.KIL, UC.QUE, and UC.PWF will usually be zero, especially for conventional drivers.

Every driver must, however, be concerned with its particular values for UC.ALG, UC.NPR, and UC.LGH.

The driver writer is 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 driver writer)

Driver access:

Initialized, not referenced.

Description:

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

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

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

The remaining bits are reserved for system use and expansion.

U.CW1 (set by driver writer)

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 characteristic words are retrieved from the UCB and placed in the requester's buffer on issuance of a GLUN$ Executive directive (Get LUN Information). It is the responsibility of the driver writer to supply the contents of these four words in the assembly source code of the driver data structure.

U.CW1 is defined as follows:

```
DV.REC  Bit 0--Record-Oriented Device(1=yes)
DV.CCL  Bit 1--Carriage-Control Device(1=yes)
DV.TTY  Bit 2--Terminal Device(1=yes)
DV.DIR  Bit 3--Directory Device(1=yes)
DV.SDI  Bit 4--Single Directory Device(1=yes)
DV.SQD  Bit 5--Sequential Device(1=yes)
DV.PSE  Bit 12--Pseudo Device(1=yes)
DV.COM  Bit 13--Device Mountable as a
                Communications Channel(1=yes)
DV.F11  Bit 14--Device mountable as a FILES-11
                device(1=Yes)
DV.MNT  Bit 15--Device mountable(1=yes)
```

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 driver writer)

Driver access:

Initialized, read-only.

Description:

Default buffer size.

U.SCB (SCB pointer)

Driver access:

Initialized, read-only.

Description:

This field contains a pointer to the SCB for this UCB. In general, R4 on entry to the driver via the driver dispatch table will contain the value in this word, since the SCB is frequently referenced by service routines.

U.ATT (initialize to zero)

Driver access:

Initialized, not referenced.

Description:

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

U.BUF (reserve two words of storage)

Driver access:

Not initialized, read/write.

Description:

U.BUF labels two consecutive words which serve as a communication region between $GTPKT and the driver. If a non-transfer function is indicated, then U.BUF, U.BUF+2, and U.CNT receive the first three parameter words from the I/O Packet.

For transfer operations, the format of these two words depends on the setting of UC.NPR in U.CTL. The driver does not format the words; all formatting is completed prior to the driver receiving 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.

Format for program transfer devices:

The format is identical to that for the second two words of I.IOSB in the I/O Packet. See section 3.1.1.1.

In general, the driver will not manipulate these words when performing I/O to program transfer device. It most likely will use the Executive routines Get Byte, Get Word, Put Byte, and Put Word to effect data transfers between the device and the user's buffer.

Format for an NPR device:

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

Word 1

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

Word 2

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

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

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

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

The details of the construction of the Address Doubleword appear in Appendix A.

U.CNT (reserve one word of storage)

Driver access:

Not initialized, read/write.

Description:

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

U.BUF and U.CNT are used to keep track of the current data item in the buffer for the current transfer (except for NPR transfers). Since this field is being altered dynamically, the I/O Packet may be needed to reissue an I/O operation.

Device-Dependent Words:

Driver access:

Not initialized, read/write.

Description:

The field is variable in length and is established by the driver writer to suit driver-specific requirements.

CHAPTER 4

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 code for the associated service.

4.1 SYSTEM-STATE REGISTER CONVENTIONS

In system state, R5 and R4 are, by convention, established as non-volatile registers. This means that an internally called routine is required to save and restore these two registers if it intends to destroy their contents. Note that drivers are entered directly from interrupts and have to call $INTSV to preserve R5 and R4.

R3, R2, R1, and R0 are volatile registers and may be used by a called routine without save and restore responsibilities.

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 and employed only when ample justification can be given to demonstrate the value added to overall system performance by virtue of the proposed departure.

4.2 SERVICE CALLS

DEVICE MESSAGE OUTPUT

$DVMSG

Device Message Output is in the 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.
;-
```

Notes:

1. 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. $FORK is called by a driver to switch from a partially interruptible level (its state following a call on $INTSV) to a fully interruptible 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. T
   fork processing routine assumes that entry conditions are set up
   $INTSV.

GET BYTE $GTBYT

Get Byte is in the file BFCTL.

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.
;-
```

GET PACKET $GTPKT

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
; 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 REQUE
; 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
; 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.
;-
```

GET WORD $GTWRD

Get Word is in the file BFCTL.

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.
;-
```

INTERRUPT SAVE $INTSV

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
; OR JUMPS TO $INTXT.
;
; INPUTS:
;
;       4(SP)=PS WORD PUSHED BY INTERRUPT.
;       2(SP)=PC WORD PUSHED BY INTERRUPT.
;       0(SP)=SAVED R5 PUSHED BY 'JSR R5,$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 LOADED AND A RETURN TO THE CALLER IS EXECUTED.
;-
```

INTERRUPT EXIT $INTXT

Interrupt Exit is in the file SYSXT.

Calling sequence:

```
    JMP    $INTXT
```

Description:

```
;+
; **-$INTXT-INTERRUPT EXIT
;
; THIS ROUTINE IS CALLED VIA A JUMP 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.
;-
```

I/O ALTERNATE ENTRY and I/O DONE

$IOALT
$IODON

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 REQUE
; 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.
;       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.
;-
```

I/O FINISH

$IOFIN

I/O Finish is in the file IOSUB. Drivers rarely call I/O Finish, but they should be aware of the fact that this routine is executed when $IOALT or $IODON is called.

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.
;-
```

PUT BYTE $PTBYT

Put Byte is in the file BFCTL.

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.
;-
```

PUT WORD $PTWRD

Put Word is in the file BFCTL.

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.
;-
```

CHAPTER 5

INCLUDING A USER-WRITTEN DRIVER - AN EXAMPLE

The example which follows is a complete illustration of the procedures required to add a driver to an RSX-11M system. The driver in the example supports the punch capability of the PC11 Paper Tape Reader/Punch.

5.1 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 a reader only (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.

5.2 DATA STRUCTURE 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 will be seen below, in a particular case, building a conventional driver is indeed a straightforward and modest undertaking.

5.2.1 The Data Structure

The data structure source is shown below and is self-explanatory. Special note should be taken of the legal function mask words, starting at line 45. The standard function codes listed in Table 3-1 were used in creating the mask. Thus, the Punch driver will accept 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 two Access/Deaccess functions are legal for FCS compatibility, but are no-op'ed. Write Virtual Block is legal but will be 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.
DEA	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(10)-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 mask words appear on lines 45 thru 52 in the data structure source.

The function code selections for record-oriented devices are intended to match FCS requirements for file-structured devices. When FCS executes an Access For Write, it will simply be marked a no-op. This tends to minimize FCS 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.

```
 1          .TITLE USRTB
 2          .IDENT /01/
 3
 4 ;
 5 ; COPYRIGHT 1975, 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 ; J. PASCUSNIK  25-NOV-74
22 ;
23 ; CONTROL BLOCKS FOR PAPER TAPE PUNCH DRIVER
24 ;
25 ; MACRO LIBRARY CALLS
26 ;
27
28          .MCALL DEVDF$,HWDDF$
29          DEVDF$                  ;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 COVE
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.
```

* Appendix B lists all macros which exist in RSX-11M and generate control offsets.

```
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
62          .BYTE   0,0             ;PHYSICAL UNIT NUMBER, UNIT STATUS EXTENSION
63          .WORD   DV.REC          ;FIRST DEVICE CHARACTERISTICS WORD
64                                  ;   (RECORD-ORIENTED DEVICE)
65          .WORD   0               ;SECOND DEVICE CHARACTERISTICS WORD
66                                  ;   (FOR INTERNAL USE BY DRIVER)
67          .WORD   0               ;THIRD DEVICE CHARACTERISTICS WORD
68                                  ;   (FOR INTERNAL USE BY DRIVER)
69          .WORD   64.             ;FOURTH DEVICE CHARACTERISTICS WORD
70                                  ;   (DEFAULT BUFFER SIZE IN BYTES)
71          .WORD   PPSCB           ;POINTER TO SCB
72          .WORD   0               ;TCB ADDRESS OF ATTACHED TASK
73          .BLKW   1               ;RELOCATION BIAS OF BUFFER OF CURRENT
74                                  ;   I/O REQUEST
75          .BLKW   1               ;ADDRESS OF BUFFER OF CURRENT I/O REQUEST
76          .BLKW   1               ;BYTE COUNT OF CURRENT I/O REQUEST
77 PPND=.
78 ;
79 ; PAPER TAPE PUNCH INTERRUPT VECTOR
80 ;
81          .ASECT
82 .=74
83          .WORD   $PPINT          ;ADDRESS OF INTERRUPT ROUTINE
84          .WORD   PR7!0           ;INTERRUPT AT PRIORITY 7 (CONTROLLER=0)
85          .PSECT
86 ;
87 ; PAPER TAPE PUNCH STATUS CONTROL BLOCK
88 ;
89 PPSCB:  .WORD   0               ;CONTROLLER I/O QUEUE LISTHEAD
90                                  ;   (POINTER TO FIRST ENTRY)
91          .WORD   .-2             ;   (POINTER TO LAST ENTRY)
92          .BYTE   PR4,74/4        ;DEVICE PRI, INTERRUPT VECTOR ADDRESS/4
93          .BYTE   0,4             ;CURRENT AND INITIAL TIMEOUT COUNTS
94          .BYTE   0,0             ;CONTROLLER INDEX AND STATUS
95                                  ;   (0=IDLE, 1=BUSY)
96          .WORD   177554          ;ADDRESS OF CONTROL STATUS REGISTER
97          .BLKW   1               ;ADDRESS OF CURRENT I/O PACKET
98          .BLKW   4               ;FORK BLOCK ALLOCATION
99
00          .END
```

5.2.2 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. A few preliminary comments will simplify the examination of the code itself.

Since the PP driver is a DEC product and will eventually be part of a released system, conditionalized sections in the code exist that will be included or deleted based on answers provided by the user to the configuration queries posed during system generation. The system generation questions determine the value of a symbol defined in the assembly prefix file RSXMC.MAC. The conditionalized sections of code are then controlled by the value of the symbol. The conditionalized code appears in multi-controller drivers and is recommended for all driver implementations.

Conditionalized code for PP is implemented as follows:

P$$P11 is set to the number of controllers the driver is to service. This sets the size of CNTBL and conditionally creates 'TEMP', if P$$P11 >1. Also, if P$$P11 >1, code is generated to save PS in TEMP for retrieval on return from $INTSV, and the controller number is decoded from the low-order four bits of the saved PS and used to index into CNTBL to obtain the UCB address. For P$$P11=1, CNTBL is one word long, TEMP is not necessary, and the UCB address is always the first entry in CNTBL.

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

Initiator;

Power Failure;

Interrupt;

Timeout, and

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 via device timeout, as is the device-not-ready condition.

The PP driver uses the following Executive services:

$INTSV
$INTXT
$GTPKT
$GTBYT
$DVMSG

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

The code contained in lines 128-130 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 (i.e., out of paper tape) and the system has generated the detach for the aborting process.

```
  1          .TITLE PPDRV
  2          .IDENT /01/
  3
  4 ;
  5 ; COPYRIGHT 1975, DIGITAL EQUIPMENT CORP., MAYNARD, MASS.  01754
  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 ; J. PASCUSNIK 25-NOV-74
 22 ;
 23 ; PC11 PAPER TAPE PUNCH DRIVER
 24 ;
 25 ; MACRO LIBRARY CALLS
 26 ;
 27
 28          .MCALL  ABODF$,DEVDF$,HWDDF$,PKTDF$,TCBDF$
 29          ABODF$                  ;DEFINE TASK ABORT CODES
 30          DEVDF$                  ;DEFINE DEVICE CONTROL BLOCK OFFSETS
 31          HWDDF$                  ;DEFINE HARDWARE REGISTER SYMBOLS
 32          PKTDF$                  ;DEFINE I/O PACKET OFFSETS
 33          TCBDF$                  ;DEFINE TASK CONTROL BLOCK OFFSETS
 34
 35 ;
 36 ; EQUATED SYMBOLS
 37 ;
 38 ; PAPER TAPE PUNCH STATUS WORD BIT DEFINITIONS (U.CW2)
 39 ;
 40
 41 WAIT=100000                      ;WAITING FOR DEVICE TO COME ON-LINE (1=YES)
 42 ABORT=40000                      ;ABORT CURRENT I/O REQUEST (1=YES)
 43 TRAIL=200                        ;CURRENTLY PUNCHING TRAILER (1=YES)
 44
 45 ;
 46 ; LOCAL DATA
 47 ;
 48 ; CONTROLLER IMPURE DATA TABLES (INDEXED BY CONTROLLER NUMBER)
 49 ;
 50
 51 CNTBL:    .BLKW   P$$P11          ;ADDRESS OF UNIT CONTROL BLOCK
 52
 53
 54          .IF GT  P$$P11-1
 55
 56 TEMP:     .BLKW   1               ;TEMPORARY STORAGE FOR CONTROLLER NUMBER
```

```
57
58          .IFTF
59
60 ;
61 ; DRIVER DISPATCH TABLE
62 ;
63
64 $PPTBL:: .WORD   PPINI           ;DEVICE INITIATOR ENTRY POINT
65          .WORD   PPCAN           ;CANCEL I/O OPERATION ENTRY POINT
66          .WORD   PPOUT           ;DEVICE TIMEOUT ENTRY POINT
67          .WORD   PPPWF           ;POWERFAIL ENTRY POINT
68
69 ;+
70 ; **-PPINI-PC11 PAPER TAPE PUNCH CONTROLLER INITIATOR
71 ;
72 ; THIS ROUTINE IS ENTERED FROM THE QUEUE I/O DIRECTIVE WHEN AN I/O REQUEST
73 ; IS QUEUED AND AT THE END OF A PREVIOUS I/O OPERATION TO PROPAGATE THE EXECU
74 ; TION OF THE DRIVER. IF THE SPECIFIED CONTROLLER IS NOT BUSY, THEN AN ATTEMP
75 ; IS MADE TO DEQUEUE THE NEXT I/O REQUEST. ELSE A RETURN TO THE CALLER IS
76 ; EXECUTED. IF THE DEQUEUE ATTEMPT IS SUCCESSFUL, THEN THE NEXT I/O OPER-
77 ; ATION IS INITIATED. A RETURN TO THE CALLER IS THEN EXECUTED.
78 ;
79 ; INPUTS:
80 ;
81 ;       R5=ADDRESS OF THE UCB OF THE CONTROLLER TO BE INITIATED.
82 ;
83 ; OUTPUTS:
84 ;
85 ;       IF THE SPECIFIED CONTROLLER IS NOT BUSY AND AN I/O REQUEST IS WAIT-
86 ;       ING TO BE PROCESSED, THEN THE REQUEST IS DEQUEUED AND THE I/O OPER-
87 ;       ATION IS INITIATED.
88 ;-
89
90          .ENABL  LSB
91 PPINI:   CALL    $GTPKT          ;GET AN I/O PACKET TO PROCESS
92          BCS     PPPWF           ;IF CS CONTROLLER BUSY OR NO REQUEST
93
94 ;
95 ; THE FOLLOWING ARGUMENTS ARE RETURNED BY $GTPKT:
96 ;
97 ;       R1=ADDRESS OF THE I/O REQUEST PACKET.
98 ;       R2=PHYSICAL UNIT NUMBER OF THE REQUEST UCB.
99 ;       R3=CONTROLLER INDEX.
100 ;      R4=ADDRESS OF THE STATUS CONTROL BLOCK.
101 ;      R5=ADDRESS OF THE UCB OF THE CONTROLLER TO BE INITIATED.
102 ;
103 ; PAPER TAPE PUNCH I/O REQUEST PACKET FORMAT:
104 ;
105 ;       WD. 00 -- I/O QUEUE THREAD WORD.
106 ;       WD. 01 -- REQUEST PRIORITY, EVENT FLAG NUMBER.
107 ;       WD. 02 -- ADDRESS OF THE TCB OF THE REQUESTER TASK.
108 ;       WD. 03 -- POINTER TO SECOND LUN WORD IN REQUESTER TASK HEADER.
109 ;       WD. 04 -- CONTENTS OF THE FIRST LUN WORD IN REQUESTER TASK HEADER (U
110 ;       WD. 05 -- I/O FUNCTION CODE (IO.WLB, IO.ATT OR IO.DET).
111 ;       WD. 06 -- VIRTUAL ADDRESS OF I/O STATUS BLOCK.
112 ;       WD. 07 -- RELOCATION BIAS OF I/O STATUS BLOCK.
113 ;       WD. 10 -- I/O STATUS BLOCK ADDRESS (REAL OR DISPLACEMENT + 140000).
114 ;       WD. 11 -- VIRTUAL ADDRESS OF AST SERVICE ROUTINE.
115 ;       WD. 12 -- RELOCATION BIAS OF I/O BUFFER.
116 ;       WD. 13 -- BUFFER ADDRESS OF I/O TRANSFER.
117 ;       WD. 14 -- NUMBER OF BYTES TO BE TRANSFERRED.
```

```
118 ;         WD. 15 -- NOT USED.
119 ;         WD. 16 -- NOT USED.
120 ;         WD. 17 -- NOT USED.
121 ;         WD. 20 -- NOT USED.
122 ;
123
124           MOV     R5,CNTBL(R3)     ;SAVE UCB POINTER FOR INTERRUPT ROUTINE
125           CLR     U.CW2(R5)        ;CLEAR ALL SWITCHES
126           CMPB    I.FCN+1(R1),#IO.WLB/256. ;WRITE LOGICAL BLOCK FUNCTION?
127           BEQ     10$              ;IF EQ YES
128           MOV     I.TCB(R1),R0     ;GET REQUESTER TCB ADDRESS
129           BITB    #TS.ABO,T.STAT+2(R0) ;TASK BEING ABORTED?
130           BNE     65$              ;IF NE YES - DON'T PUNCH TRAILER
131           BIS     #TRAIL,U.CW2(R5) ;OTHERWISE FUNCTION IS ATTACH OR DETACH
132                                    ;    SET FLAG TO PUNCH TRAILER
133           MOV     #170.,U.CNT(R5) ;SET COUNT FOR 170 NULLS
134 10$:      BIS     #WAIT,U.CW2(R5) ;ASSUME WAIT FOR DEVICE OFF LINE
135           TST     @S.CSR(R4)       ;DEVICE OFF LINE?
136           BMI     80$              ;IF MI YES
137 20$:      BIC     #WAIT,U.CW2(R5) ;DEVICE ON LINE, CLEAR WAIT CONDITION
138           MOVB    S.ITM(R4),S.CTM(R4) ;SET TIMEOUT COUNT
139           MOV     #100,@S.CSR(R4) ;ENABLE INTERRUPTS
140
141 ;
142 ; POWERFAIL IS HANDLED VIA THE DEVICE TIMEOUT FACILITY AND THEREFORE CAUSES
143 ; NO IMMEDIATE ACTION ON THE DEVICE. THIS IS DONE TO AVOID A RACE CONDITION
144 ; THAT COULD EXIST IN RESTARTING THE I/O OPERATION
145 ;
146
147 PPPWF:  RETURN                     ;
148
149 ;+
150 ; **-$PPINT-PC11 PAPER TAPE PUNCH CONTROLLER INTERRUPTS
151 ;-
152
153 $PPINT::                           ;;;REF LABEL
154
155           .IFT
156
157           MOV     PS,TEMP          ;;;SAVE CONTROLLER NUMBER
158
159           .IFTF
160
161           CALL    $INTSV,PR4       ;;;SAVE REGISTERS AND SET PRIORITY
162
163           .IFT
164
165           MOV     TEMP,R4          ;;;RETRIEVE CONTROLLER NUMBER
166           BIC     #177760,R4       ;;;CLEAR ALL BUT CONTROLLER NUMBER
167           ASL     R4               ;;;CONVERT TO CONTROLLER INDEX
168           MOV     CNTBL(R4),R5     ;;;RETRIEVE ADDRESS OF UCB
169
170           .IFF
171
172           MOV     CNTBL,R5         ;;;RETRIEVE ADDRESS OF UCB
173
174           .ENDC
175
176
177           MOV     U.SCB(R5),R4     ;;;GET ADDRESS OF STATUS CONTROL BLOCK
178           MOVB    S.ITM(R4),S.CTM(R4) ;;;RESET TIMEOUT COUNT
```

```
179             MOV     S.CSR(R4),R4    ;;;POINT R4 TO CONTROL STATUS REGISTER
180             MOV     (R4)+,U.CW3(R5) ;;;SAVE STATUS
181             BMI     60$             ;;;IF MI, ERROR
182             SUB     #1,U.CNT(R5)    ;;;DECREMENT CHARACTER COUNT
183             BCS     50$             ;;;IF CS, THEN DONE
184             TSTB    U.CW2(R5)       ;;;CURRENTLY PUNCHING TRAILER?
185             BPL     30$             ;;;IF PL NO
186             CLRB    (R4)            ;;;PUNCH A NULL
187             BR      40$             ;;;BRANCH TO EXIT FROM INTERRUPT
188 30$:        CALL    $GTBYT          ;;;GET NEXT BYTE FROM USER BUFFER
189             MOVB    (SP)+,(R4)      ;;;LOAD BYTE IN OUTPUT REGISTER
190 40$:        JMP     $INTXT          ;;;EXIT FROM INTERRUPT
191 50$:        INC     U.CNT(R5)       ;;;RESET BYTE COUNT
192 60$:        CLR     -(R4)           ;;;DISABLE PUNCH INTERRUPTS
193             CALL    $FORK           ;;;CREATE SYSTEM PROCESS
194             MOV     U.SCB(R5),R4    ;POINT R4 TO SCB
195             MOV     S.PKT(R4),R1    ;POINT R1 TO I/O PACKET
196             MOV     I.PRM+4(R1),R1  ;   AND PICK UP CHARACTER COUNT
197             SUB     U.CNT(R5),R1    ;CALCULATE CHARACTERS TRANSFERRED
198             MOV     #IS.SUC&377,R0  ;ASSUME SUCCESSFUL TRANSFER
199             TST     U.CW3(R5)       ;DEVICE ERROR?
200             BPL     70$             ;IF PL NO
201 65$:        MOV     #IE.VER&377,R0  ;UNRECOVERABLE HARDWARE ERROR CODE
202 70$:        CALL    $IODON          ;INITIATE I/O COMPLETION
203             BR      PPINI           ;BRANCH BACK FOR NEXT REQUEST
204
205 ;
206 ; DEVICE TIMEOUT RESULTS IN A NOT READY MESSAGE BEING PUT OUT 4 TIMES A
207 ; MINUTE. TIMEOUTS ARE CAUSED BY POWERFAILURE AND PUNCH FAULT CONDITION
208 ;
209
210 PPOUT:      CLRB    @S.CSR(R4)      ;;;DISABLE PUNCH INTERRUPT
211             CLRB    PS              ;;;ALLOW INTERRUPTS
212 80$:        MOV     #IE.DNR&377,R0  ;ASSUME DEVICE NOT READY ERROR
213             MOV     U.CW2(R5),R1    ;ARE WE WAITING FOR DEVICE READY?
214             BPL     70$             ;IF PL NO, TERMINATE I/O REQUEST
215             MOV     #IE.ABO&377,R0  ;ASSUME REQUEST IS TO BE ABORTED
216             ASL     R1              ;ABORT REQUEST?
217             BMI     70$             ;IF MI YES
218             TST     @S.CSR(R4)      ;PUNCH READY?
219             BPL     20$             ;IF PL YES
220             MOV     #T.NDNR,R0      ;SET FOR NOT READY MESSAGE
221             MOVB    #1,S.CTM(R4)    ;SET TIMEOUT FOR 1 SECOND
222             DECB    S.STS(R4)       ;TIME TO OUTPUT MESSAGE?
223             BNE     PPPWF           ;IF NE NO
224             MOVB    #15.,S.STS(R4)  ;SET TO OUTPUT NEXT MESSAGE IN 15. SEC
225             CALLR   $DVMSG          ;OUTPUT MESSAGE
226             .DSABL  LSB
227 ;
228
229 ; CANCEL I/O OPERATION-FORCE I/O TO COMPLETE IF DEVICE IS NOT READY
230 ;
231
232 PPCAN:      CMP     R1,I.TCB(R0)    ;;;REQUEST FOR CURRENT TASK?
233             BNE     10$             ;;;IF NE NO
234             BIS     #ABORT,U.CW2(R5) ;;;SET FOR ABORT IF DEVICE NOT READY
235 10$:        RETURN                  ;;;
236
237             .END
```

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 248K 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 was necessary for internal representation of an address until it was actually used in an I/O operation.

One choice may have been to carry about the hardware virtual address. This, however, was rejected since lengthy conversions are involved, especially when the user for whom the address was being manipulated was not presently mapped into the memory management registers. Additionally, a scheme was needed whereby the mapped/unmapped characteristic of a given system would be relatively transparent to device drivers.

The choice was made to encode two words as the internal representation of a physical address, and to transform virtual addresses for I/O operations into the internal doubleword format.

A.2 CREATING THE ADDRESS DOUBLEWORD

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

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

The virtual address in the DPB is structured as follows:

Bits 0-5	Displacement in 32-word block
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. This 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 six is placed in bits 13-15 to enable use of PAR #6, which will be used by the Executive to service I/O for program transfer devices.

For non-program transfer (NPR) devices, the driver requirements for manipulating the address doubleword are direct and are discussed with the description of U.BUF in section 3.1.4.1.

APPENDIX B
SYSTEM DATA STRUCTURES AND SYMBOLIC DEFINITIONS

```
        .MACRO  ABODFS,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  ABODFS
        .ENDM
        .ENDM

        .MACRO  CLKDFS,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:
;-
```

```
,MRKT='B'0                          ;MARK TIME REQUEST
,SCHD='B'2                          ;TASK REQUEST WITH PERIODIC RESCHEDULING
,SSHT='B'4                          ;SINGLE SHOT TASK REQUEST
,SYST='B'6                          ;SINGLE SHOT INTERNAL SYSTEM SUBROUTINE (IDENT)
,SYTK='B'8.                         ;SINGLE SHOT INTERNAL SYSTEM SUBROUTINE (TASK)
```

CLOCK QUEUE CONTROL BLOCK TYPE INDEPENDENT OFFSET DEFINTIONS

```
        .ASECT
=0
,LNK:'L' .BLKW 1                    ;CLOCK QUEUE THREAD WORD
,RQT:'L' .BLKB 1                    ;REQUEST TYPE
,EFN:'L' .BLKB 1                    ;EVENT FLAG NUMBER (MARK TIME ONLY)
,TCB:'L' .BLKW 1                    ;TCB ADDRESS OR SYSTEM SUBROUTINE IDENTIFICATION
,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
,AST:'L' .BLKW 1                    ;AST ADDRESS
,SRC:'L' .BLKW 1                    ;FLAG MASK WORD FOR 'BIS' SOURCE
,DST:'L' .BLKW 1                    ;ADDRESS OF 'BIS' DESTINATION
```

CLOCK QUEUE CONTROL BLOCK-PERIODIC RESCHEDULING DEPENDENT OFFSET DEFINITIONS

```
=C.TIM+4                            ;START OF DEPENDENT AREA
,RSI:'L' .BLKW 2                    ;RESCHEDULE INTERVAL IN CLOCK TICKS
,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
,SUB:'L' .BLKW 1                    ;SUBROUTINE ADDRESS
        .BLKW   2                   ;TWO UNUSED WORDS
,LGTH='B'.                          ;LENGTH OF CLOCK QUEUE CONTROL BLOCK
        .PSECT

        .MACRO  CLKDFS
        .ENDM
        .ENDM
```

```
        .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, THE
; THERE WOULD BE TWO DCB'S, ONE FOR ALL DL11-A INTERFACED TELETYPES, AND ONE
; FOR ALL DH11 INTERFACED TELETYPES. A SIMILAR SITUATION WOULD ARISE IF A
; SYSTEM CONTAINED TWO RK11 DISK CONTROLLERS, ONE DCB WOULD BE REQUIRED
; FOR EACH CONTROLLER.
;-

        .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.

        .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
```

```
        .MACRO  F11DFS
;
; VOLUME CONTROL BLOCK
;
        .ASECT
.=0
V.TRCT: .BLKW   1               ;TRANSACTION COUNT
V.IFWI: .BLKW   1               ;INDEX FILE WINDOW
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
V.VOWN: .BLKW   1               ;VOLUME OWNER'S UIC
V.VPRO: .BLKW   1               ;VOLUME PROTECTION
V.VCHA: .BLKW   1               ;VOLUME CHARACTERISTICS
V.FPRO: .BLKW   1               ;VOLUME DEFAULT FILE PROTECTION
V.VFSQ: .BLKW   1               ;VOLUME FILE SEQUENCE NUMBER
V.FRBK: .BLKB   1               ;NUMBER OF FREE BLOCKS ON VOLUME HIGH BYTE
V.LRUC: .BLKB   1               ;COUNT OF AVAILABLE LRU SLOTS IN FCB LIST
        .BLKW   1               ;NUMBER OF FREE BLOCKS ON VOLUME LOW BITS
V.LGTH:                         ;SIZE IN BYTES OF VCB

;
; FILE CONTROL BLOCK
;
        .ASECT
.=0
F.LINK: .BLKW   1               ;FCB CHAIN POINTER
F.FNUM: .BLKW   1               ;FILE NUMBER
F.FSEQ: .BLKW   1               ;FILE SEQUENCE NUMBER
        .BLKW   1               ;UNUSED
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.STRK=.-F.LBN          ;SIZE OF STATICS BLOCK
F.STAT: .BLKW   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
        .BLKW   1               ;UNUSED
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.BPS=100000           ;BYPASS ACCESS INTERLOCK IF SET
W.VBN:  .BLKB   1               ;HIGH BYTE OF 1ST VBN MAPPED BY WINDOW
W.WISZ: .BLKB   1               ;SIZE IN RTRV PTRS OF WINDOW (7 BITS)
        .BLKW   1               ;LOW ORDER WORD OF 1ST VBN MAPPED
W.FCB:  .BLKW   1               ;FILE CONTROL BLOCK ADDRESS
W.RTRV:                         ;OFFSET TO 1ST RETRIEVAL POINTER IN WINDOW

        .PSECT
        .MACRO  F11DF$
        .ENDM   F11DF$
        .ENDM   F11DF$

        .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.PCBT:'L'.BLKW 4               ;TASK PARTITION DESCRIPTOR
H.PCBC:'L'.BLKW 3*4             ;COMMON PARTITION DESCRIPTORS
        .BLKW   1               ;BOUNDRY WORD FOR PCB ADDRESSES (ALWAYS=0)
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.RSVD:'L'.BLKW 1               ;RESERVED POINTER LOCATION
H.EFLM:'L'.BLKW 4               ;EVENT FLAG MASK WORDS
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
        .BLKW   1               ;RESERVED WORD
H.FPSA:'L'.BLKW 1               ;POINTER TO FLOATING POINT/EAE SAVE AREA
        .BLKW   1               ;RESERVED WORD
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
        .PSECT

        .MACRO  HDRDF$
        .ENDM
        .ENDM
```

```
        .MACRO HWDDFS,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
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

        .MACRO  HWDDF$
        .ENDM
        .ENDM

        .IIF NDF S$$YDF , .LIST



        .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
```

```
        .MACRO  PCBDF$

;+
; PARTITION CONTROL BLOCK OFFSET DEFINITIONS
;-

        .ASECT
.=0
P.LNK:  .BLKW   1               ;LINK TO NEXT PARTITION PCB
P.NAM:  .BLKW   2               ;PARTITION NAME IN RAD50
P.SUB:  .BLKW   1               ;POINTER TO NEXT SUBPARTITION
P.MAIN: .BLKW   1               ;POINTER TO MAIN PARTITION
P.REL:  .BLKW   1               ;STARTING PHYSICAL ADDRESS OF PARTITION
P.SIZE: .BLKW   1               ;SIZE OF PARTITION IN BYTES
P.BLKS:                         ;SIZE OF PARTITION IN 32W BLOCKS (SYSTEM ONLY)
P.WAIT: .BLKW   1               ;PARTITION WAIT QUEUE LISTHEAD (2 WORDS)
P.SWSZ: .BLKW   1               ;PARTITION SWAP SIZE (SYSTEM ONLY)
P.BUSY: .BLKB   2               ;PARTITION BUSY FLAGS
P.TCB:  .BLKW   1               ;TCB ADDRESS OF OWNER TASK
P.NAPR: .BLKB   1               ;NUMBER OF APR'S TO LOAD
P.STAT: .BLKB   1               ;PARTITION STATUS FLAGS


        .IF DF  M$$MGE

P.PDR:  .BLKW   1               ;CONTENTS OF LAST PDR TO BE LOADED
P.HDR:  .BLKW   1               ;POINTER TO HEADER CONTROL BLOCK

        .IFF

P.HDR=P.REL                     ;POINTER TO HEADER CONTROL BLOCK

        .ENDC


P.LGTH=.                        ;LENGTH OF PARTITION CONTROL BLOCK
        .PSECT

;+
; PARTITION STATUS BYTE BIT DEFINITIONS
;-

PS.COM=200                      ;LIBRARY OR COMMON BLOCK (1=YES)
PS.PIC=100                      ;POSITION INDEPENDENT LIBRARY OR COMMON (1=YES)
PS.SYS=40                       ;SYSTEM CONTROLLED PARTITION (1=YES)
PS.DRV=20                       ;DRIVER IS LOADED IN PARTITION (1=YES)
PS.APR=7                        ;STARTING APR NUMBER MASK

        .MACRO  PCBDF$
        .ENDM
        .ENDM
```

```
        .MACRO  PKTDF$,L,B

;+
; ASYNCHRONOUS SYSTEM TRAP CONTROL BLOCK OFFSET DEFINITIONS
;-

        .ASECT
.=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                        ;VIRTUAL ADDRESS OF I/O STATUS BLOC
        .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
I.LGTH='.',                     ;LENGTH OF I/O REQUEST CONTROL BLOCK
        .PSECT

        .MACRO  PKTDF$
        .ENDM
        .ENDM
```

```
   .MACRO  SCBDF$,L,B


TUS CONTROL BLOCK

 STATUS CONTROL BLOCK (SCB) DEFINES THE STATUS OF A DEVICE CONTROLLER.
RE IS ONE SCB FOR EACH CONTROLLER IN A SYSTEM. THE SCB IS POINTED TO
JNIT CONTROL BLOCKS. TO EXPAND ON THE TELETYPE EXAMPLE ABOVE, EACH TELE-
: INTERFACED VIA A DL11-A WOULD HAVE A SCB SINCE EACH DL11-A IS AN IN-
:NDENT INTERFACE UNIT. THE TELETYPES INTERFACED VIA THE DH11 WOULD ALSO
 HAVE AN SCB SINCE THE DH11 IS A SINGLE CONTROLLER BUT MULTIPLEXES MANY
TS IN PARALLEL.


   .ASECT
'72
[$'L' .BLKB 1                   ;NUMBER OF REGISTERS TO COPY ON ERROR
:$'L' .BLKB 1                   ;OFFSET TO FIRST DEVICE REGISTER
/$'L' .BLKW 1                   ;SAVED I/O ACTIVE BITMAP AND POINTER TO EMB
($'L' .BLKW 1                   ;DEVICE I/O ACTIVE BIT MASK
:'L' .BLKW 2                    ;CONTROLLER I/O QUEUE LISTHEAD
:'L' .BLKB 1                    ;DEVICE PRIORITY
:'L' .BLKB 1                    ;INTERRUPT VECTOR ADDRESS /4
:'L' .BLKB 1                    ;CURRENT TIMEOUT COUNT
:'L' .BLKB 1                    ;INITIAL TIMEOUT COUNT
:'L' .BLKB 1                    ;CONTROLLER INDEX
:'L' .BLKB 1                    ;CONTROLLER STATUS (0=IDLE,1=BUSY)
:'L' .BLKW 1                    ;ADDRESS OF CONTROL STATUS REGISTER
:'L' .BLKW 1                    ;ADDRESS OF CURRENT I/O PACKET
:'L' .BLKW 1                    ;FORK/TIME REQUEST BLOCK LINK WORD
   .BLKW   1                    ;FORK-PC/TIME-QUEUE REQUEST TYPE
   .BLKW   1                    ;FORK-R5/TIME-REQUEST IDENTIFICATION
   .BLKW   1                    ;FORK-R4/TIME-LOW ORDER TIME
:'L' .BLKW 1                    ;FORK-UNUSED/TIME-HIGH ORDER TIME/CHANNEL CONTROL BLOCK
   .BLKW   1                    ;FORK-UNUSED/TIME-SUBROUTINE ADDRESS
:'L' .BLKW 6                    ;11/70 EXTENDED MEMORY UNIBUS DEVICE C-BLOCK

   .PSECT


'US CONTROL BLOCK PRIORITY BYTE CONDITION CODE STATUS BIT DEFINITIONS


:='B'1                          ;ERROR IN PROGRESS (1=YES)
:='B'2                          ;ERROR LOGGING ENABLED (0=YES)
:='B'4                          ;ERROR LOGGING AVAILABLE (1=YES)
:12                             ;SPARE BIT

   .MACRO  SCBDF$,X,Y
   .ENDM
   .END
```

```
        .MACRO  TCBDF$,L,B

;+
; 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.TCB:'L' .BLKW 1               ;POINTER TO T.LNK OF TCB ITSELF
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' .BLKB 3              ;TASK STATUS WORD AND EXTENSION BYTE
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.LGTH='B'.                     ;LENGTH OF TASK CONTROL BLOCK
T.EXT='B'.-T.LGTH               ;LENGTH OF TCB EXTENSION
        .PSECT

;+
; TASK STATUS DEFINITIONS
;
; TASK STATUS WORD
;-

TS.EXE='B'100000                ;TASK IS IN EXECUTION (0='B'YES)
TS.RDN='B'40000                 ;I/O RUN DOWN IN PROGRESS (1='B'YES)
TS.DST='B'20000                 ;AST RECOGNITION DISABLED (1='B'YES)
TS.MSG='B'10000                 ;ABORT MESSAGE BEING OUTPUT (1='B'YES)
TS.PMD='B'4000                  ;DUMP TASK ON SYNCHRONOUS ABORT (0=YES)
TS.STP='B'2000                  ;TASK STOPPED FOR TERMINAL INPUT (1=YES)
TS.REM='B'1000                  ;REMOVE TASK ON EXIT (1='B'YES)
TS.ACP='B'400                   ;ANCILLARY CONTROL PROCESSOR (1='B'YES)
TS.AST='B'200                   ;AST IN PROGRESS (1='B'YES)
TS.CHK='B'100                   ;TASK IS CHECKPOINTABLE (0='B'YES)
TS.BFX='B'40                    ;TASK BEING FIXED IN MEMORY (1='B'YES)
TS.FXD='B'20                    ;TASK FIXED IN MEMORY (1='B'YES)
TS.OUT='B'10                    ;TASK IS OUT OF MEMORY (1='B'YES)
TS.CKP='B'4                     ;TASK IS CHECKPOINTED (1='B'YES)
TS.CKR='B'2                     ;CHECKPOINT REQUESTED (1='B'YES)
TS.CKD='B'1                     ;CHECKPOINT DISABLED (1='B'YES)

;+
; TASK BLOCKING STATUS MASK
;-

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

;+
; TASK STATUS BYTE EXTENSION
;-
```

```
S.HLT='B'200                    ;TASK IS BEING HALTED (1='B'YES)
S.PRV='B'100                    ;TASK IS PRIVILEGED (1='B'YES)
S.ABO='B'40                     ;TASK MARKED FOR ABORT (1='B'YES)
S.MCR='B'20                     ;TASK REQUESTED AS EXTERNAL MCR FUNCTION (1='B'YES)
S.SPN='B'10                     ;SAVED TS.SPN ON AST IN PROGRESS
S.SPN='B'4                      ;TASK SUSPENDED (1='B'YES)
S.WFR='B'2                      ;SAVED TS.WFR ON AST IN PROGRESS
S.WFR='B'1                      ;TASK IN WAITFOR STATE (1='B'YES)

        .MACRO  TCBDF$
        .ENDM
        .ENDM


        .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
=0
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.SWL='B'4000                  ;UNIT SOFTWARE WRITE LOCKED (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 DH11/DJ11 (1=YES)
U2.DJ1='B'40000                 ;UNIT IS A DJ11 (1=YES)
U2.RMT='B'20000                 ;UNIT IS REMOTE (1=YES)
U2.LOG='B'400                   ;USER LOGGED ON TERMINAL (0=YES)
U2.LWC='B'200                   ;LOWER CASE TO UPPER CASE CONVERSION (1=YES
U2.OFF='B'100                   ;OUTPUT IS TURNED OFF (1=YES)
U2.PND='B'40                    ;OUTPUT BYTE PENDING (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.SLV='B'1                     ;UNIT IS A SLAVE TERMINAL (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='B'YES)
US.FOR='B'40                    ;UNIT IS MOUNTED AS FOREIGN VOLUME (1=YES)
US.MDM='B'20                    ;UNIT IS MARKED FOR DISMOUNT (1=YES)

;+
; UNIT STATUS EXTENSION BIT DEFINITIONS
;-

US.OFL='B'1                     ;UNIT OFFLINE (1=YES)
US.RED='B'2                     ;UNIT REDIRECTABLE (0=YES)

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

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

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

US.WCK='B'10                    ;WRITE CHECK ENABLED (1=YES)

;+
; 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)

        .MACRO  UCBDF$,X,Y
        .ENDM
        .ENDM
```

INDEX

RSX-11M Guide to Writing an I/O Driver
DEC-11-OMWDA-B-D

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form

Did you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

_____

_____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

_____

_____

_____

_____

_____

_____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

If you require a written reply, please check here. ☐

Please cut along this line.

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

INESS REPLY MAIL
POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

age will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

digital equipment corporation

Printed in U.S.A.