Professional™

300 series

Guide to Writing a P/OS I/O Driver and Advanced Programmer's Notes

Developer's Tool Kit

digital

software

Guide to Writing a P/OS I/O Driver and Advanced Programmer's Notes

Order No. AA-BT73B-TH
Order No. AD-BT73BA-T1

June 1987

This reference manual describes the procedures for writing an I/O driver for P/OS systems. Executive routine descriptions and sample code are included. Advanced programmer information is also provided.

REQUIRED SOFTWARE:	Host Tool Kit V3.0, or PRO/Tool Kit V3.2
OPERATING SYSTEM:	P/OS V3.2

DIGITAL EQUIPMENT CORPORATION
Maynard, Massachusetts 01754-2571

First Printing, April 1982
Revised, December 1985
Updated, June 1987

Update Notice Number 1

Guide to Writing a P/OS I/O Driver and Advanced Programmer's Notes

AD-BT73B-T1

June 1987

Insert this page in the Guide to Writing a P/OS I/O Driver and Advanced Programmer's Notes to maintain an up-to-date record of changes to the manual.

NEW AND CHANGED INFORMATION

This update reflects software changes and additions made in P/OS Version 3.2. Also included are corrections to the original documentation.

INSTRUCTIONS

Add the following pages to the Guide to Writing a P/OS I/O Driver and Advanced Programmer's Notes as replacements for, or additions to, the current pages. The technical changes made on replacement pages are indicated in the left margin by change bars. Changes of an editorial nature are not marked.

OLD PAGE	NEW PAGE
Title page/copyright page	Title page/copyright page
iii/iv through xi/xii	iii/iv through xvii/xviii
4-55/4-56	4-55/4-56
4-63/4-64 through 4-65/4-66	4-63/4-64 through 4-65/4-66
5-3/5-4 through 5-5/5-6	5-3/5-4 through 5-5/5-6
(Continued)	

OLD PAGE	NEW PAGE
7-45/7-46 through 7-65/7-66	7-45/7-46 through 7-65/7-66
C-1/C-2 through C-33/blank	C-1/C-2 through C-35/C-36
D-1/D-2	D-1/D-2
E-1/E-2 through E-43/E-44	E-1/E-2 through E-45/E-46
Index-1 through Index-6	Index-1 through Index-8
Reader's comments/mailer	Reader's comments/mailer

CONTENTS

CHAPTER 1 **P/OS I/O DRIVERS**

CHAPTER 2 **DEVICE DRIVER I/O STRUCTURES**

CHAPTER 3 EXECUTIVE SERVICES AND DRIVER PROCESSING

CHAPTER 4 PROGRAMMING SPECIFICS FOR WRITING AN I/O DRIVER

CHAPTER 5 INCORPORATING A USER-SUPPLIED DRIVER INTO P/OS

CHAPTER 6 DEBUGGING A USER-SUPPLIED DRIVER

CHAPTER 7 EXECUTIVE SERVICES AVAILABLE TO AN I/O DRIVER

CHAPTER 8 HANDLING SPECIAL USER BUFFERS

CHAPTER 9 ACCESSING VIDEO HARDWARE AND TERMINAL SUBSYSTEM

APPENDIX A P/OS SYSTEM DATA STRUCTURES AND SYMBOLIC DEFINITIONS

APPENDIX B TASK BUILDING AND CLUSTER LIBRARIES

APPENDIX C **FILES-11 ON-DISK STRUCTURE SPECIFICATION**

APPENDIX D FILES-11 QIO INTERFACE TO THE ACPS

APPENDIX E **QUAD SERIAL LINE UNIT (PC3XC-BA)**

FIGURES

TABLES

PREFACE

Document Objectives

The primary goal of this manual is to introduce P/OS physical I/O concepts, define Executive and I/O service routine protocol, describe system I/O data structures, and prescribe I/O service routine coding procedures. This information is in sufficient detail to allow you to:

- Prepare software that interfaces with the Executive and supports a conventional I/O device.
- Incorporate the user-written software into an P/OS system.
- Detect typical errors that cause the system to crash.
- Use Executive service routines that an I/O service routine typically employs.
- Develop P/OS video applications that directly access the video hardware.

CAUTION

Unless explicitly noted otherwise, all information in this manual is subject to change without notice.

Intended Audience

This manual is written for the senior-level system programmer who is familiar with the hardware characteristics of both the Professional 300 Series and the device that the user-written software supports. The programmer should also be knowledgeable about DIGITAL peripheral devices and experienced in using the software supplied with a P/OS system. The manual neither describes general Executive concepts nor defines general system structures. The manual does describe I/O concepts, the Executive role in processing I/O requests, and some pertinent aspects of I/O processing done by DIGITAL-supplied software. Therefore, with a firm understanding of hardware characteristics and P/OS system software, a senior-level system programmer could attempt to write an I/O driver.

Structure of This Document

This manual has three types of information: conceptual, procedural, and reference. The following are abstracts of the chapters in the document:

- **Chapter 1**, *P/OS I/O Drivers*, introduces terms and concepts fundamental to understanding physical I/O in P/OS, and describes the protocol that a driver must follow to preserve system integrity. It summarizes advanced driver features and P/OS capabilities helpful in becoming acquainted with overall Executive and driver interaction.

- **Chapter 2**, *Device Driver I/O Structures*, continues the conceptual discussion begun in Chapter 1. It introduces on a general level the software data structures involved in handling I/O operations at the device level, examines typical arrangements of data structures that are necessary for controlling hardware functions, and presents a macroscopic software configuration that summarizes the logical relationships of the I/O data structures.

- **Chapter 3**, *Executive Services and Driver Processing*, ends the conceptual presentation. It summarizes how an I/O request originates, how the Executive processes the request, and how a driver would use Executive services to satisfy an I/O request.

- **Chapter 4**, *Programming Specifics for Writing an I/O Driver*, provides the detailed reference information necessary to code a conventional I/O driver. Included is a summary of programming standards and protocol, an introduction to the programming facilities and requirements for both the driver data base itself and the executable code that constitutes the driver, and an extensive elaboration of the driver data base and of the driver code.

- **Chapter 5**, *Incorporating A User-Supplied Driver into P/OS*, supplies the procedural information that you need to assemble and build a loadable driver image, load it into memory, and make accessible the devices that the driver supports.

- **Chapter 6,** *Debugging A User-Supplied Driver,* summarizes software features provided to help you uncover faults in drivers and gives procedures to follow that might prove successful in isolating faults in drivers.

- **Chapter 7,** *Executive Services Available to An I/O Driver,* gives general coding information relating to the PDP-11 and P/OS Executive service routines.

- **Chapter 8,** *Sample Driver Code,* shows the source code for the data base and driver of a conventional device and an excerpt of source code from a driver that handles special user buffers.

- **Chapter 9,** *Accessing Video Hardware and Terminal Subsystem,* provides reference information on the driver's control of Professional video hardware and software.

- **Appendix A,** *System Data Structures and Symbol Definitions,* lists the source code of system macro calls that define system device structures, driver-related structures, and system-wide symbolic offsets needed to access those structures.

- **Appendix B,** *Task Building and Cluster Libraries,* is a collection of three documents describing overlaying task structures, cluster libraries, and task organization.

- **Appendix C,** *Files-11 On-Disk Structure Specification,* describes the general-purpose file structure intended for use on medium and large-size PDP-11 systems.

- **Appendix D,** *QIO Interface to the ACPs,* describes the QIO level interface to the file processors (ACPs).

- **Appendix E,** *Quad Serial Line Unit Driver (PC3XC-BA),* describes the Quad Asynchronous Communications Module (PC3XC-BA).

Associated Documents

Included in your P/OS Tool Kit documentation are documents that describe both the software and hardware on the system. The software documents are listed and described in the *Tool Kit User's Guide.* Consult this document for concise summaries of software-related publications. For information on hardware technical specifications, see the *Professional 300 Series Technical Manual.*

Also, it is recommended that you refer to the P/OS Executive listings, which are published on microfiche. It is entitled *Executive Listings and Maps*.

Associated Files

As mentioned in your installation guide, the directory [ZZPRIVDEV] on the PRODCL2 diskette contains several library and symbol table files, which are needed for writing privileged applications.

CHAPTER 1

P/OS I/O DRIVERS

Device drivers on P/OS are the primary method of interfacing the Executive's I/O subsystem with hardware attached to the computer. Most DIGITAL-supplied hardware is supported by drivers accompanying the system that the user receives. This chapter introduces the concept of device drivers and explains driver operations and features.

1.1 VECTORS AND CONTROL AND STATUS REGISTERS

Associated with a device contoller are device control and status registers. The addresses of these registers are determined by the physical slot in which the controller has been inserted, rather than the actual option module. A given controller may have up to 64. words of device registers as shown in Table 1-1. To provide a unique identification for controllers, a hardware ID is expected to be present, and may be accessed at the first address within a given slot's device register address range. The bootstrap and diagnostic ROM examines each option slot and places the hardware IDs in a configuration table located at the top of physical memory. This table is referenced by PROLOD to resolve the hardware ID specified in the controller table (CTB) located in the driver's database. The table may also be accessed by the executive's WIMP$ directive (See the *P/OS System Reference Manual* for further details of the directive arguments and table format.) Certain controllers are physically present on the motherboard. These devices have predefined device registers and are fully described in the *Professional 300 Series Technical Manual*.

Table 1-1: Option Slot Address Assignments

Physical Slot Position	Logical Slot Number	Device Register Address Range	Vector Address Interrupt A ICSRA= 17773206	Vector Address Interrupt B ICSRB= ICSRA+4
1	0	17774000-17774177	300	304
2	1	17774200-17774377	310	314
3	2	17774400-17774577	320	324
4	3	17774600-17774777	330	334
5	4	17775000-17775177	340	344
6	5	17775200-17775377	350	354

System Peripheral Address Assignments

Logical Slot Number	Device Register Address Range	(ICSR= 17773202) Vector Address	Device Type
0			Not used
1	17773500-17773506	200	Keyboard receiver
2		204	Keyboard transmitter
3	17773300-17773314	210	Comm. Port rec./trans.
4		214	Comm. Port modem status
5	17773400-17773406	220	Printer Port receiver
6		224	Printer Port trans.
7	17773000-17773032	230	System clock

Optionally, a controller may utilize one or two of the two-word areas associated with each slot, called interrupt vectors. A vector provides a connection between the device and the software that services the device. A vector allows a device to trigger certain software actions because of some external condition related to the device. When a device interrupts, the vector address is sent to the processor. The first word of the interrupt vector contains the address of the interrupt service routine for that device. The processor uses the second word of the vector as a new Processor Status Word. Thus, when the processor services the interrupt, the first word of the vector is taken as the new Program Counter (PC) and the second word is the new PS.

Space is reserved on the PDP-11 for the interrupt vectors. This space is in the low part of Kernel I-space. The vectors are considered to be in Kernel mode virtual address space and are thus mapped by the Executive. Because the interrupt vector is in Kernel space, the Executive receives control of the processor on every interrupt.

1.2 SERVICE ROUTINES

The service routine that is entered to process an interrupt is most frequently in the device driver. Device drivers vary in complexity depending on the capabilities of the type of device and the number of device units they service.

Although linked into the Executive structures, a driver resides in memory outside the virtual address space of the Executive. An application can add or remove a driver by means of PROLOD, a callable POSSUM system routine. In addition, any driver not required for a period of time need not be loaded. The space normally occupied by the unloaded driver can hold user tasks or another driver.

1.2.1 Executive and Driver Layout

A device driver is a logical extension of the Executive that is not contiguous in physical memory with the Executive code. Active Page Registers (APRs)* 0 through 4 map the Executive, APR 7 is reserved to map the I/O page, and APR 5 maps the driver. Therefore, a driver is by default restricted to the 4K words of space mapped by APR 5 unless it controls its own mapping with APR 6 to gain access to an extra 4K words.

The virtual to physical mapping of a P/OS system is shown in Figure 1-1.

---

* Active Page Register is a term referring to the Memory Management register pair (Page Address Register (PAR) and Page Descriptor Register (PDR).) Refer to the *Professional 300 Series Technical Manual* for information on hardware mapping and memory management. Refer to the *RSX11M-Plus and Micro/RSX Task Builder Manual* for a description of mapping and APR assignments by software.

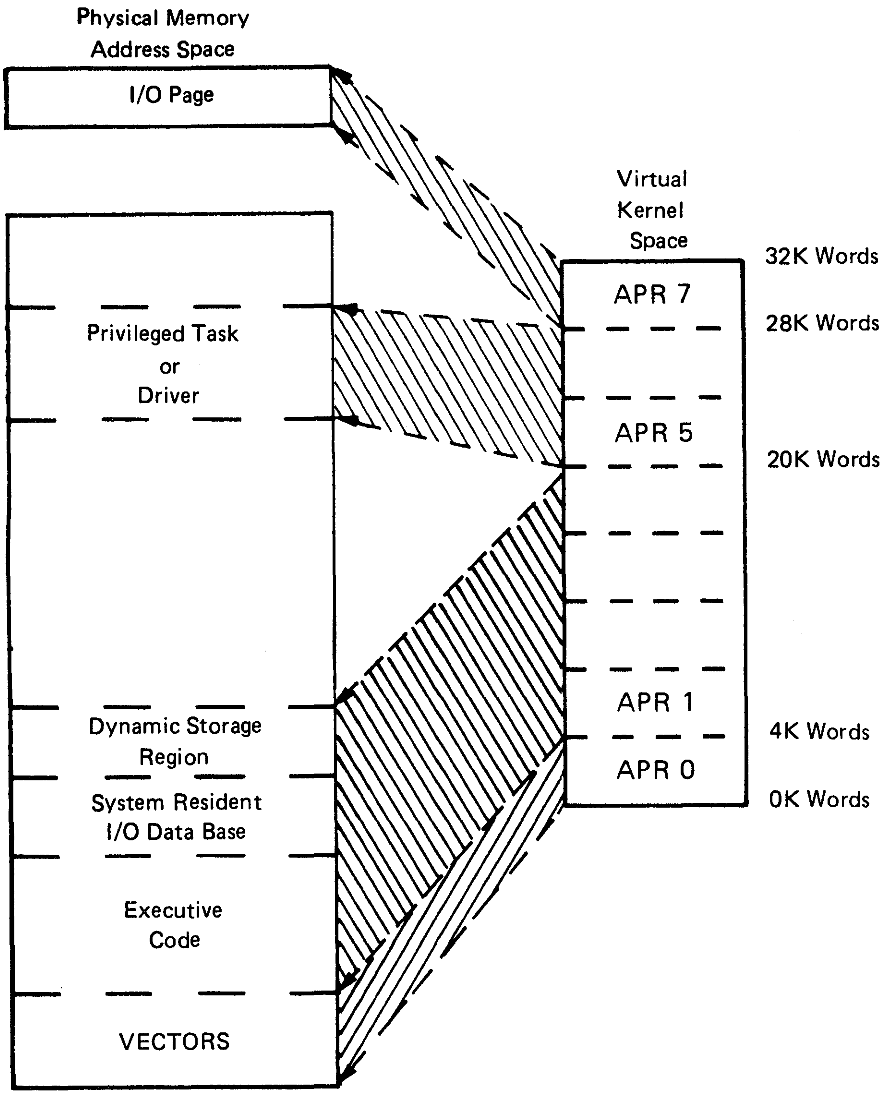


Figure 1-1: Virtual to Physical Mapping for the Executive

Virtual addresses 20K through 28K words (APR 5 and APR 6) are reserved to map drivers and privileged tasks in Kernel mode. (Although APR5 and APR6 are reserved for drivers, the Executive maps only APR5 when it calls a driver.) Finally, virtual addresses 28K through 32K words (APR 7) map the I/O page.

Thus, a device driver is mapped with the Executive code and the I/O page. When a driver has control, it can access the device registers in the I/O page to perform its operations. While in **system** state, a driver has access to all the Executive service routines to help it process I/O requests. While in **interrupt** state, the driver must save and restore APR 6 (access to APR 6 is unrestricted when the driver is in system state).

Because of the layout of the Executive and device drivers, many common functions related to I/O are centralized in the Executive as service routines. This commonality eliminates the inclusion of repetitive coding in each and every driver. Coding in each driver is therefore reduced to handling the specific functions of the device supported.

1.2.2 Driver Contents

A device driver consists of two parts. One part is the executable instructions of the driver itself. This part has the entry points to the driver. The entry points are those places where the Executive calls the driver to perform a specific action, and their addresses are established in the driver dispatch table (DDT). The table contains addresses of routines in a fixed order so that the Executive can enter the driver at the appropriate place for a given action.

The other part of a device driver is the data structures forming the data base that describes the controllers and units supported by the driver. Two structures, the controller table (CTB) and the controller request block (KRB), describe the controller of the device being supported. Because the CTB supplies generic information about the controller type, only one CTB need exist for each controller type on a system. The KRB holds information related to a specific controller and therefore each controller has its associated KRB.

Three structures in the driver data base--the device control block (DCB), the unit control block (UCB), and the status control block (SCB)--describe the device as a logical entity. The DCB contains information related to the type of device, whereas the UCB holds information specific to an individual unit of the device. The SCB is used mainly to store data (driver context) concerning an operation in progress on the device unit.

The driver data structures are tailored to the number of controllers on the system, the number of units attached to each controller, and the types of features the devices support. The structures increase in complexity as the number of supported features increases.

1.3 EXECUTIVE AND DRIVER INTERACTION

The Executive and a driver interact by accessing and manipulating common data structures. An I/O activity typically begins when a task generates a request for input or output. The Executive performs preliminary processing of that request before it initiates the driver. This preliminary processing, called predriver initiation, is common for all drivers and eliminates a great deal of code from all drivers.

In performing predriver initiation, the Executive accesses the driver data structures to assess the legality of the I/O request. For example, cells in the device control block (DCB) define the functions that the driver supports. If the function specified in the I/O request is not supported by the driver, the Executive need not call the driver. The driver is not aware of the I/O request. Therefore, the Executive calls the driver only when the predriver initiation warrants it.

1.3.1 The Driver Process

When the Executive does call the driver to process an I/O request, the driver begins I/O initiation. Once an I/O request is created, a driver process is initiated. The Executive has queued to the driver an I/O packet that must be processed to satisfy the request. Potentially there exist on the system as many driver processes as there are distinct units capable of being active simultaneously. (Moreover, some drivers supporting advanced features can have multiple I/O requests simultaneously active for a given unit. In this case, each active I/O request is part of a separate driver process. Refer to Section 1.4.3 for more information.

Central to a full understanding of a driver and the I/O structure is the difference between a driver process and the driver code. The driver code, (which is pure instruction), invokes an Executive routine called $GTPKT to get an I/O packet to process. This activity generates data for the request being processed and the unit doing the processing. The driver process, once initiated, starts the proper I/O function, waits for a completion interrupt and performs any required data transfers. It then completes the I/O by specifying I/O status and requesting another I/O packet. This sequence of execution steps continues until the I/O queue is empty and the driver process terminates.

Because a driver may be capable of servicing several I/O requests in parallel, it is possible that, for a single driver, many driver processes exist at the same time. However, there is only one copy of driver code. The driver process is reentrant code and the data that defines the state of the code is stored in the driver data base when the process is not executing (for example, when it is waiting for an interrupt). The driver process executes driver code for a particular device type on behalf of a specific unit. If independent units of a particular device type are concurrently active, several driver processes are also active at the same time, each with its own set of data.

1.3.2 Interrupt Dispatching and the Interrupt Control Block

Once a driver starts an I/O function, it must await the I/O completion interrupt. When a device interrupt occurs, the processor pushes the current PS and PC onto the current stack and loads the new PS and PC from the device controller interrupt vector. By convention, the PS in the interrupt vector is preset with a priority of 7 and the number of the controller associated with the vector. (The controller number, which identifies a particular controller for a given controller type, is in the low-order four bits.)

For a driver, the hardware cannot dispatch directly to the interrupt service routine in the driver because the driver is mapped outside the address space of the Executive. Therefore, some code in the Executive must initially handle the interrupt, load the mapping context of the driver, and dispatch to the proper driver. This code resides in the Executive in a structure called an interrupt control block (ICB). Figure 1-2 shows this mechanism. A common Executive coroutine, called interrupt save ($INTSI) is called from the ICB. The $INTSI coroutine saves two registers, R4 and R5, which are thereafter free for the driver to use. These registers are typically used by drivers to hold addresses of the data blocks containing unit status and control information, the SCB and UCB. (Most Executive routines assume these two registers hold pointers to the two structures. If the driver needs to use more registers, it saves them on the stack and restores them when it finishes.) Kernel APR 5 is then saved and the driver is mapped through APR 5 and called at the interrupt entry point. When the interrupt save coroutine returns to the driver, the driver runs at the interrupt level of the device that it is servicing and has two free registers that it can use.

The driver may then run for a short interval at the partially interruptable level. By convention, this interval should not exceed 500 microseconds. When the driver finishes processing the interrupt, it may execute a RETURN instruction to transfer control back to the coroutine which gives control of the CPU to the next process.*

---

* An Executive interrupt exit routine, $INTXT, exists to standardize the way a driver exits from an interrupt. This routine is executed by the $INTSI coroutine. Therefore, interrupt exit processing is effected via the "RTS PC" (RETURN) instruction.

Thus, the ICB actually contains a JSR instruction to an Executive interrupt save routine ($INTSI) that performs the following:

- Save R4 and R5
- Save the Kernel mapping (APR 5)
- Load APR 5 to map the driver
- Transfer control to the driver via a JSR instruction
- Restore the mapping after return from the driver
- Perform interrupt exit processing
- Restore R4 and R5
- Return from interrupt

Thus, the interrupt vector for a controller serviced by a driver points to an ICB rather than to the driver.

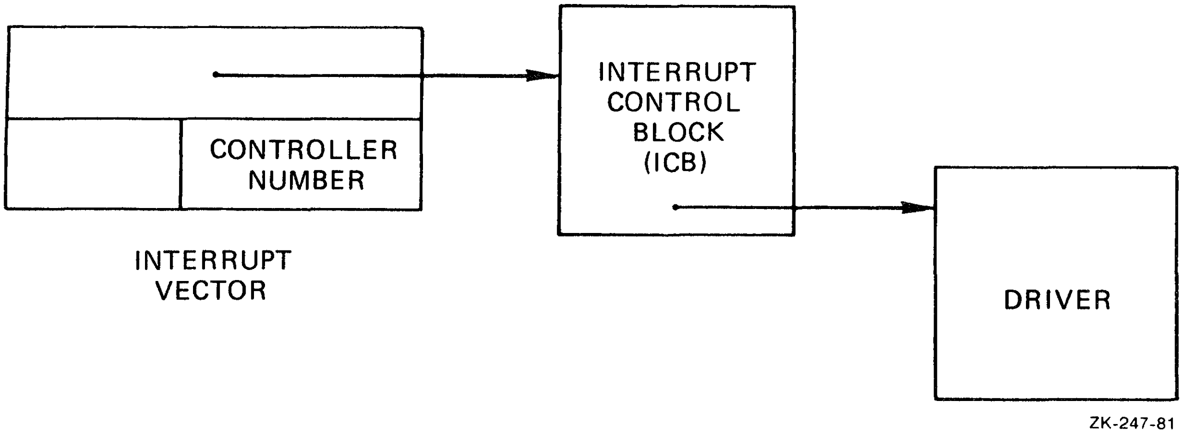


Figure 1-2: Interrupt Dispatching for a Driver

The ICB conceptually allows up to 128 controllers of the same type on a system. The low-order four bits in the PS of the interrupt vector restricts the number of controllers to 16. In the ICB, the system maintains a controller group number and the PS bits describe the controller number within the group. To obtain the controller index (controller number #2), the executive interrupt service routine first multiplies the controller within group number that was in the low 4 bits of the PS by two. The group number byte in the ICB is then added to this number.

The simplest case in handling an interrupt is that in which a controller can have only one unit active at any one time. Multiple controllers may be active concurrently, yet only one unit per controller may be active. The interrupt service routine in the driver uses the controller index passed in R4 to index a table in the CTB and to access the proper KRB. From the KRB, the UCB (via K.OWN) may be determined to access the proper unit data and context. For those devices that have a static one-to-one static relationship between a controller and a perticular UCB, the controller index may be used to index a table of UCB addresses within the driver.

The more complex case in dispatching an interrupt is that in which a controller can have multiple units operating in parallel. This is an advanced driver feature called overlapped seek I/O and is described in Section 1.4.1.

1.3.3 Interrupt Servicing and Fork Process

A driver handling an interrupt and operating at the partially interruptable level may need to (1) access structures in its data base or (2) call centralized Executive service routines which may access structures in the data base. Because a driver may have more than one process active simultaneously, the driver itself may need to access structures in the data base shared among separate, unrelated processes. A method must exist to coordinate access to the data structures shared among the processes and the Executive.

The mechanism that coordinates access to the shared structures is called the fork process. An Executive routine, called fork ($FORK), causes the driver process to be placed in a queue of processes waiting for access to the shared data structures, to run at processor priority level 0, and to be completely interruptable.*

A driver must therefore call the fork routine before it calls any other Executive service routine (except for $INTSI), or before it accesses any device-specific (nonprivate) structures in its data base. If a driver does not follow this protocol, it will corrupt the system data base and lead to a system crash.

A driver that calls the fork routine requests the Executive to transform it into a fork process. The routine saves a snapshot of the

---

* By convention, drivers may operate at a partially interruptable level for no more than 500 microseconds. Some drivers conceivably could need more time than this convention allows. Thus, an additional reason for the fork mechanism is to preserve the response time of the system and not lock out interrupts from lower-priority levels.

process in a fork block. The snapshot is the context of the driver process--the PC of the process and the contents of R4 and R5. The fork block itself can (and usually does) reside in the I/O data structure holding the status information of the device being serviced (that is, the status control block, or SCB). The Executive maintains a list of fork blocks in FIFO order. A new fork block is added to the list after the last block in the list.

When the driver calls $FORK, the CPU priority is lowered to 0, which allows other interrupts to be serviced. When there are no more pending interrupts (they have either been dismissed or the drivers have called $FORK), the Executive checks to see whether the first interrupt preempted a priority 0 Executive process. If a preemption occurred, the Executive process is continued from where it was interrupted.* If no priority level 0 Executive process was interrupted, the Executive executes the process at the head of the fork list. The Executive restores the saved context of the process from the SCB and returns control to the driver at the statement immediately following the call to the fork routine. The process is unaware that a pause of indeterminate length has elapsed.

Fork processes thereby are granted FIFO access to the common I/O data structures. Once granted such access, a fork process has control of the structures until it exits. The protocol guarantees that the driver process has unrestricted access to shared system data structures. As one fork process exits, the next in the list is eligible to run and access the data structures. Thus, the fork mechanism allows both controlled access to the common data structures and sufficient time to process an interrupt without locking up the system.

The status of a fork process lies between an interrupting routine and a task requesting system resources. This is known as "system state." Interrupt routines are run first and can be interrupted only by higher-priority interrupts. Processes in the fork list run after other system processes either terminate or call $FORK themselves. Because system processes save and restore user task registers and cannot be interrupted by a fork process, a fork process can use all registers. The fork processes are completely interruptable. Tasks run only when the fork list is empty.

The fork mechanism establishes linear, or serial, access to the shared data structures. For example, an Executive routine that completes I/O processing ($IODON) manipulates the I/O queue to deallocate an I/O packet that the driver processed. If multiple processes were allowed

---

* The stack must be restored to its state on interrupt entry before calling $FORK. Therefore, it cannot be used to pass additional context.

to alter the queue at random times, the queue pointers could become disarranged. Without the fork mechanism, any process could be interrupted by a higher-priority process and not be able to complete its manipulation. Because the Executive completes a currently active fork process before it starts the next fork process in the queue, the integrity of the I/O data structures is maintained if all routines that call $IODON run at system state.

Between the time that a driver process calls $FORK and the Executive starts the process at system state, the driver cannot call $FORK again for that same device. If the $FORK routine is called again before the first process starts, context stored in the fork block for the first fork process is overwritten. However, once a fork process starts, the data in the fork block is stale and the process may call $FORK again while it is at system state. If the driver does not ensure against unexpected interrupts, it may double fork as described above. As a result of the double fork, the system will bugcheck (crash) with an IOT trap as a result of a failed sanity check while queuing the forkblock. A common protocol used by DIGITAL device drivers is to clear the saved PC in the forkblock immediately following a fork, and to test this work before forking. If the word is non-zero, it is not possible to call $FORK.

If all drivers adhere to the interrupt protocol, the integrity of the I/O data structures is preserved. Thus, when a device interrupt occurs while a fork process is executing, the protocol demands that the service routine handling the interrupt not destroy any of the registers. The registers are part of the context of the fork process. After the driver dismisses the interrupt or itself becomes a fork process, the interrupted fork process can safely resume execution with its proper context. If any driver violates the protocol, the integrity of the I/O data structures is endangered. (That is, the system crashes in mysterious ways.)

1.3.4 Nonsense Interrupt Entry Points

All vectors for off-line devices and vectors for which there are no devices contain the addresses of Executive nonsense interrupt entry points. Code at these special entry points exists to properly dismiss unexpected interrupts from these devices via an RT1 instruction.

1.4 ADVANCED DRIVER FEATURES

This section introduces optional features so you can better understand the structures and concepts described in the remainder of the manual.

1.4.1 Overlapped Seek I/O

Some disk devices allow multiple device units attached to the same controller to execute operations in parallel. This is called overlapped seek support and is a software option designed to take advantage of a hardware feature found in most advanced disk drives and controllers. This feature allows any or all drives to be attached to the same controller, allowing this functionality to execute a seek function simultaneously. Each unit may perform a seek operation independent of what another unit may be doing. Only one data transfer can occur at any one time. Some types of drives allow seek functions to overlap a data transfer function, whereas other types do not.

The increased difficulty for overlapped seek devices stems from determining whether the controller or the unit generated the interrrupt. Most control functions issued to the drive unit (including the positioning commands SEEK and SEARCH) terminate with a unit interrupt. The controller reports the physical unit number of the interrupting unit. A controller interrupt indicates the termination of a function (usually a data transfer command) that changes the controller status from busy to ready. Only one unit may issue a data transfer complete notification to a particular controller at any one time because only one data transfer can be in progress at any one time. Most hardware defers seek termination interrupts until the current data transfer is complete.

To handle interrupts for a device that supports overlapped seek operations, a device controller-specific interrupt service routine must be built into the driver to examine the device registers in order to determine whether the interrupt was initiated by the controller or the drive unit. Using the controller index on interrupt entry, the routine uses this as an offset into a table of addresses in a structure (called the controller table or CTB) in the I/O data base. The routine accesses the table to determine the address of the I/O data structure of the interrupt controller (called the controller request block or KRB) that generated the interrupt. Accessing the KRB yields the address of the CSR of that controller and having the CSR address allows the routine to examine the device registers.

If the controller itself initiated the interrupt, the routine determines the data base structure of the unit that is active. This determination is possible because such a controller interrupt relates to a termination of a data transfer, and only one such unit can be active for a data transfer. A cell in the KRB has the address of the data structure describing the active unit (the unit control block or UCB). The routine can then determine the address of the driver dispatch table and transfer control to the driver.

If a device unit initiated the interrupt, the routine retrieves its unit number from the device registers. Using the physical unit number, the routine indexes a table at the end of the KRB to yield the address of the related UCB. The driver is entered through the driver dispatch table.

1.4.2 Delayed Controller Access

Drivers that support overlapped seeks also must request access to a controller before executing a function on an independent unit and must release access after completing the function. To take maximum advantage of simultaneous operation of units on one controller, the system delays controller access when the controller is busy.

The Executive maintains a request queue for the controller. Whenever a driver process requests access to a controller and must wait for access to the controller, the Executive places the associated fork block in the controller request queue. When a driver releases a controller, the Executive automatically grants access to the next driver process waiting for access. Precedence is given to positioning requests over requests for data transfer. The controller request queue thereby provides the means for the Executive to synchronize access.

1.4.3 Full Duplex Input/Output

In certain circumstances it may be necessary for a driver to handle more than one I/O request on a unit at the same time. Typically a driver processes only one I/O packet per unit at any one time. In normal operation the driver calls the Executive routine $GTPKT to get an I/O packet to process. When $GTPKT returns an I/O packet, it marks the device busy and does not allow additional I/O until the first I/O activity completes. Therefore, only one I/O process can be in progress at the same time on a device. Full duplex operation allows more than one I/O process to be in progress on a device at the same time.

To allow full duplex operation, the $GTPKT routine has a special entry point called $GSPKT. A driver calling $GSPKT specifies an acceptance routine, to which $GSPKT returns control when an eligible packet is found. The acceptance routine determines whether to accept or reject the packet. The criteria that the acceptance routine applies could be that a write request is accepted if a write has just completed or that a read request is accepted if a read has just completed. If the routine rejects the packet, it indicates so to $GSPKT, which continues to search for another packet. If the acceptance routine accepts the packet, $GSPKT dequeues the packet and passes it to the driver but does not modify U.BUF and U.CNT in the unit control block (UCB) nor does it mark the device busy. As a result, during full duplex operation the device appears idle even while it is processing an I/O request. For this reason, it may be difficult to make use of the executive's standard driver timeout facility. Clock queue entries are suggested.

To complete an I/O request under full duplex operation, the driver calls the $IOFIN routine rather than the $IOALT or $IODON routine. $IOFIN does final processing without making the device look idle, as $IOALT and $IODON attempt to do. In full duplex operation, a unit will always appear idle to the system and the driver acceptance routine will determine whether the device can handle an I/O request.

A driver handling full duplex operations requires augmented data base structures. The conventional data base structures are defined for only one I/O request in progress per unit. Because the driver has to keep more information concerning a unit that allows two I/O requests in progress, you may have to alter the UCB and other data base structures to provide additional offsets. The DIGITAL-supplied full duplex terminal driver not only uses a lengthened UCB and a nonstandard SCB, but also connects to a dynamically allocated UCB extension.

1.4.4 Buffered Input and Output

Typically, data for input and output requests are transferred directly to and from task memory. To allow the successful transfer of data, the task cannot be checkpointed until the transfer is complete. For most high-speed devices, the transfer occurs quickly enough so that a task does not occupy memory for too long a time. For slow-speed devices, however, some mechanism must be available to avoid binding memory to a task for too long a time while the task is performing I/O.

Using the routines $TSTBF, $INIBF, and $QUEBF in the Executive module IOSUB, a driver can execute an I/O request for a slow-speed device and allow the task to be checkpointed while the request is in progress. To perform the I/O request, the driver buffers the data in memory allocated to the driver while the task is checkpointed and the I/O request is in progress.

To test whether a task is in a proper state to initiate I/O buffering, the driver calls the $TSTBF routine and passes it the address of the I/O packet. By extracting the address of the task control block (TCB) from the I/O packet, $TSTBF can examine various task attributes. For example, if the task is not checkpointable, buffered I/O is not desirable. $TSTBF returns to the driver and indicates whether buffered I/O can be performed.

If buffered I/O can be performed, the driver performs two operations. First, it establishes the buffering conditions. For an output request, it copies the task buffers to dynamically allocated pool space. For an input request, it allocates sufficient pool space to receive the incoming data. Second, the driver calls the $INIBF routine to initiate the I/O buffering. $INIBF decrements the task I/O count, increments the task's buffered I/O count in T.TIO, and releases the task for checkpointing and shuffling. If the task is currently blocked, the task state is transformed into a "stopfor" state until the task is unblocked, buffered I/O completes, or both. Checkpointing the task is subject to the normal requirements of an active or "stopfor" state as described in the *P/OS System Reference Manual*.

After the driver transfers the data, it calls the $QUEBF routine to queue the buffered I/O for completion. $QUEBF sets up a kernel asynchronous system trap (AST) for the buffered I/O request and if necessary, unstops the task. When the task is active again, a routine ($FINBF) in the Executive module SYSXT notices the outstanding AST and processes it. (If the request is for input, the routine copies the buffered data to task memory.) This mechanism occurs transparently to the task, thus the name kernel AST. The routine then calls the driver to deallocate the buffer from pool. $IOFIN completes the processing.

1.5 OVERVIEW OF INCORPORATING A USER-WRITTEN DRIVER INTO P/OS

A callable system service called PROLOD, located in the POSSUM library, is responsible for loading and unloading a driver. When loading, PROLOD establishes the linkage between the data base structures in the system device tables and the driver code being loaded. When unloading, PROLOD removes the driver's code from memory (although PROLOD removes a driver, it does not remove a data base).

To incorporate a user-written driver into P/OS, you first create two modules, one in which you define the data base and the other in which you include the driver code itself. You must supply in your code the global symbols that PROLOD needs.

PROLOD also loads the driver's data base. It reads the driver symbol definition file to find the start and end of the data base in the driver image. (Thus, you must have defined its start and end in the data base source code.) Knowing the start and end, PROLOD reads the data base from the driver image. It then places the data base in the system pool so that it resides in Executive address space, accordingly relocates pointers and links within the data base to be valid Executive addresses, and also connects the CTB and DCB(s) in the data base to the system device tables. Moreover, so that the system device tables are not corrupted by an incorrect data base, PROLOD performs many consistency and validity checks on the data base being loaded.

OVERVIEW OF INCORPORATING A USER-WRITTEN DRIVER INTO P/OS

You must build both of the following:

1. A single image (a .TSK file without a header) containing the driver code module followed by the driver data base module (see Section 5.1.3).
2. A symbol definition (.STB) file that PROLOD requires to find critical data base and driver locations.

You will link the driver image to the Executive version under which the driver will run. However, the driver image will be separate from the Executive image. PROLOD is responsible for loading both your driver data base and driver code, for connecting the data base to the system device tables, and for connecting your driver code to the data base.

CHAPTER 2

DEVICE DRIVER I/O STRUCTURES

This chapter deals mainly with structures at the block level, their relationship to the hardware configuration and functionality supported, and their relationships to each other. The precise description of each structure is given in Chapter 4.

2.1 I/O STRUCTURES

The main elements in the driver I/O environment essentially define the logical and physical characteristics of the supported hardware and establish the links and connections by which routines can access and manipulate driver data. The following subsections describe the control blocks that a driver data base module defines, and explain in general terms the purposes for each block.

2.1.1 Controller Table (CTB)*

A controller table defines a unique controller type on the system. A CTB must exist for each physical controller type. All controller tables are linked together, in a list, with the head of the list $CTLST in the Executive common area. The list of the controller tables is one of the threads through the system data base to provide access to all device-related data. The link in the last CTB in the list has a value of zero.

Associated with each CTB is a 2-character ASCII controller name which must be unique within the system. This unique name allows PROLOD to find the correct CTB for the controller type.

---

* Drivers that are not associated with hardware devices (such as in memory disk or pipe driver) do not need a CTB or KRB. DCBs, SCBs, and UCBs are a sufficient database.

Any user-written driver data base must have its own CTB. The user-created controller table will also be linked into the system CTB list, if necessary.

A CTB has generic status information, links, and pointers to other structures on the system. The table of KRB addresses in the CTB is the means by which the Executive handles interrupts for the controller type and dispatches to the correct driver routine.

2.1.2 Controller Request Block (KRB)*

The controller request block is the means by which the Executive maintains controller-specific or hardware-specific information and accesses the correct information for a unit which its associated controller owns. One KRB exists for each device controller of a given type in the configuration. It stores such data as the the device CSR and vector addresses, the slot number, the interrupt controller CSR address, and controller's status.

In a configuration where a device controller allows only one operation at a time, the KRB is combined with another structure called the status control block (SCB). (The SCB holds context for a unit while an operation is in progress.) Because only one access path is possible in such a configuration, unit context is always associated with the same controller. Moreover, because only one operation is possible at a time, the same context storage area can be used for all units attached to the controller. Thus, in a conventional driver operating environment, the context storage is merely an extension of the controller request block.

In a configuration where multiple operations in parallel on the same controller are possible, the controller context is separate from each independent unit context. Therefore, each unit capable of operating independently on a controller has the context of the current I/O operation stored in an SCB separate from the controller KRB. In such an operating environment, any unit can access the controller while other operations are pending, but only one unit can have access at a time. The KRB indicates which unit owns the controller for the current operation, and synchronizes access among driver processes on the same controller.

Where multiple operations in parallel are allowed on a controller, it may be necessary to delay access to the controller when it is busy. Therefore, in the KRB the Executive holds the head of a list of access requests called the controller request queue. The list contains fork

---

* See footnote on page 2-1.

blocks for driver processes awaiting controller access. The queue is the means by which the Executive serializes access to the controller.

When a controller allows parallel operations, the software must have a means of determining which of several units generated an interrupt. The KRB, therefore, contains a table of addresses which associate the controller with all the units connected to it. This table, indexed by physical unit number, must appear if the controller in question supports overlapped seek operations or multiple simultaneous data transfers for each physical unit attached to the controller (comm. multiplexers).

The KRB also holds the configuration status of the controller. If the KRB indicates that the controller is off-line, no activity can take place on any unit connected to the controller.

2.1.3 Device Control Block (DCB)

The device control block describes the static characteristics of a device type and of units associated with a certain device type. The DCB is the means of access to the driver dispatch table and thus to the driver. At least one DCB exists for each logical type of device on a system. There may be more than one DCB for a logical device type. (Note that the logical device type is not the same as the physical device type.)

A cell in each device control block forms a link in a forward-linked list, with the head of the list starting in a cell ($DEVHD) in the Executive common area. This list, as with the CTB list, is a main thread through the system data structures to device-related data. The link in the last DCB in the list has a value of zero.

The static data in the DCB gives such information as the generic device name, unit quantity and links to individual unit data, the address of the driver dispatch table, and types of I/O functions supported by the driver. Typically, the Executive QIO directive processing code and not the driver code accesses the DCB.

2.1.4 Unit Control Block (UCB)

The unit control block holds much of the static information about an individual device unit and contains a few dynamic parameters. Although unit control blocks need not be any prescribed length for different devices, all unit control blocks for the same DCB must be of equal length. (The UCB length is stored in the device control block to calculate the offset to a particular UCB in the concatenated set of UCBs described by the DCB's logical unit numbers.) This condition allows the UCB to contain varying amounts of unit- and device-independent data for different types of devices.

A UCB, one of which exists for each device unit, enables a driver to access most of the other structures in the I/O environment. A UCB provides access to most of the dynamic data associated with I/O operations. Given the address of a UCB, a driver may readily find most of the other data structures in which it is interested because the proper links exist. Because of this access information, the UCB is a key control block in the driver I/O structure.

The static data in the UCB includes pointers to other I/O structures, definitions of unit control bits which regulate directive processing, definitions of unit status bits which describe operational conditions, and definitions of unit- and device-dependent characteristics and storage cells.

Data in the UCB is accessed and modified by both the Executive and the driver.

2.1.5 Status Control Block (SCB)

The status control block holds driver context for operations on a device unit. In the SCB are stored such data as the pointer to the head of the queue of input/output requests; the link to the fork blocks queued for the unit; the fork process context; timeout, and unit status; and the address for the controller request block (KRB) representing the device controller (if the device has a controller).

The Executive accesses the SCB to set up an I/O request, to store context while a request is in progress, and to post results and status. When the driver accesses the SCB, it is usually for read access only.

If the controller itself cannot handle parallel operations, only one SCB is needed for each controller. In such a case, a controller can have only one unit processing a command at a time, and there is no need to store context for more than one unit at a time. There is also no need for a physically separate controller request block (KRB) to separate controller information from operation state information. Therefore, the driver data base contains the required KRB cells in the status control block since the KRB and SCB overlap.

If the controller allows parallel operations, there must be one SCB to store operation state information for each unit capable of operating independently on the controller. In such a configuration, a cell in each SCB points to the KRB of the controller to which the units are connected.

Certain operations, such as data transfers, could require exclusive use of a controller. A controller can be requested for this purpose using the routines located in the MDSUB module of the Executive (the microfiche distributed with the Tool Kit contains the MDSUB listing). If the device controller can support unqualified parallel operations to multiple units, the driver can use $GSPKT and $IOFIN, and can maintain its own unit busy state internally.

Being capable of *unqualified* parallel operations means that the controller can handle *any* operations in parallel.

2.2 DRIVER DISPATCH TABLE (DDT)

The driver dispatch table* contains the entry points to and the interrupt entry addresses for the driver. An entry point is the location at which the Executive calls the driver to perform a specific function. An interrupt entry address is a location to which the central processor or the Executive transfers control within the driver for servicing hardware interrupts. The pointer to the interrupt entry address resides in an interrupt control block.

Every driver has four conventional entry points as follows:

- I/O initiation
- cancel I/O
- device timeout
- device powerfail

Two more entry points are added for controller and unit on-line and off-line status changes:

- KRB status change
- UCB status change

For many devices, these status change entry points are merely a return to the Executive calling routine.

There are two additional entry points that have been added for the

---

* The DDT is not a structure in the strict sense of the word because it is defined in the instruction part of the driver code. However, because it contains addresses for dispatching code, it is included in the data structure description.

advanced driver feature of buffered I/O and terminal driver processing:

- Deallocate buffers (buffered I/O)
- Send next command (FDX TTDRV)

2.2.1 I/O Initiation

The Executive transfers control to this entry point to inform the driver that work for it is waiting to be done. To reduce work for the driver, the Executive performs predriver-initiation processing. (Predriver initiation is described in Chapter 3). If, at the end of predriver processing, the Executive has I/O packets queued for the driver, it calls the driver at this entry point.

When the driver receives control at its I/O initiation entry point, R5 contains the address of the UCB for the unit on which the request is to be processed. To establish access to the I/O packet, the driver calls an Executive routine that does either of the following*:

- It returns information concerning the dequeued packet to be processed and information needed to gain access to the unit's data structures
- It causes the driver to dismiss the initiation request, because the $GTPKT routine processed the request on behalf of the driver. (There may be no packet to process or the driver may already be busy.)

Once control is returned to a driver and there is a request to process, the driver must extract the information from the registers, establish data within the control blocks, and process the request. This means that the driver proceeds with an I/O request until it issues a command to the controller hardware, which physically initiates the I/O operation.

Typically a driver is called at this entry point after an I/O packet has been inserted into the I/O queue. However, a driver can be called before a packet is placed in the I/O queue. Refer to the description of the U.CTL control flag UC.QUE in Section 4.4.4 for information on queueing an I/O packet to the driver.

---

* The $GTPKT routine, which gets a packet for the driver to process, is described in Chapter 7.

2.2.2 Cancel I/O

To terminate an in-progress I/O operation, the system flushes the I/O queue and calls the driver at this entry. There are many situations in which a task, or the Executive (on behalf of an exiting task), must terminate I/O. When such a termination becomes necessary, a task issues an IO.KIL Executive QIO request and the Executive relays the request to the driver by calling the driver at its cancel entry point. Cancel requests are not queued and have no associated I/O packet.

The driver is responsible for checking that the I/O operation in-progress was issued from the task that is forcing the termination, and for completing or terminating the operation before returning to the caller.

Typically, a driver is called at this entry point only when an I/O operation is in progress. A driver also can be called, even if the unit specified is not busy. Refer to the description of the U.CTL control flag UC.KIL in Section 4.4.4 for information on unconditional cancelling of I/O. (For instance, the driver may need to clean up data structures created during pre-initiation processing (UC.QUE=1) and has "hidden" the I/O packet listhead in some other structure).

2.2.3 Device Timeout

When a driver initiates an I/O operation, it can establish a timeout count. If the operation fails to complete within the specified interval, the Executive notes the lapse and calls the driver at this entry point. Using this facility, a driver can wait for an interrupt but need not hang if the interrupt never occurs. Thus, no driver should ever stall on a request because a hardware failure prevented an expected interrupt from happening.

NOTE

> Extreme caution must be exercised to avoid completing an I/O request twice. It can happen once for timeout, and again when a pending forked process becomes active and completes I/O again.

2.2.4 Device Power Failure

The Executive calls the power failure entry point upon successful completion of loading.

2.2.5 Controller and Unit Status Change

Two entry points are required for configuration status changes of the controller and units. The Executive enters one entry point to put the controller on-line and take it off-line. The other entry point, called once for each unit whose status changes, is for putting units on-line and taking them off-line. The driver must show successful completion of the on-line or off-line request or the Executive will not effect the status change.

2.2.6 Device Interrupt Addresses

Control passes to a driver's interrupt service routine address when a device, previously initiated by the driver, completes an I/O operation and causes an interrupt in the central processor. A device may have associated with it more than one interrupt entry.

You specify the interrupt addresses in a block in the DDT. The arrangement is general enough to support multicontroller drivers such as the terminal driver. The block defines the address or addresses to include in the vector for the driver.

2.3 TYPICAL CONTROL RELATIONSHIPS

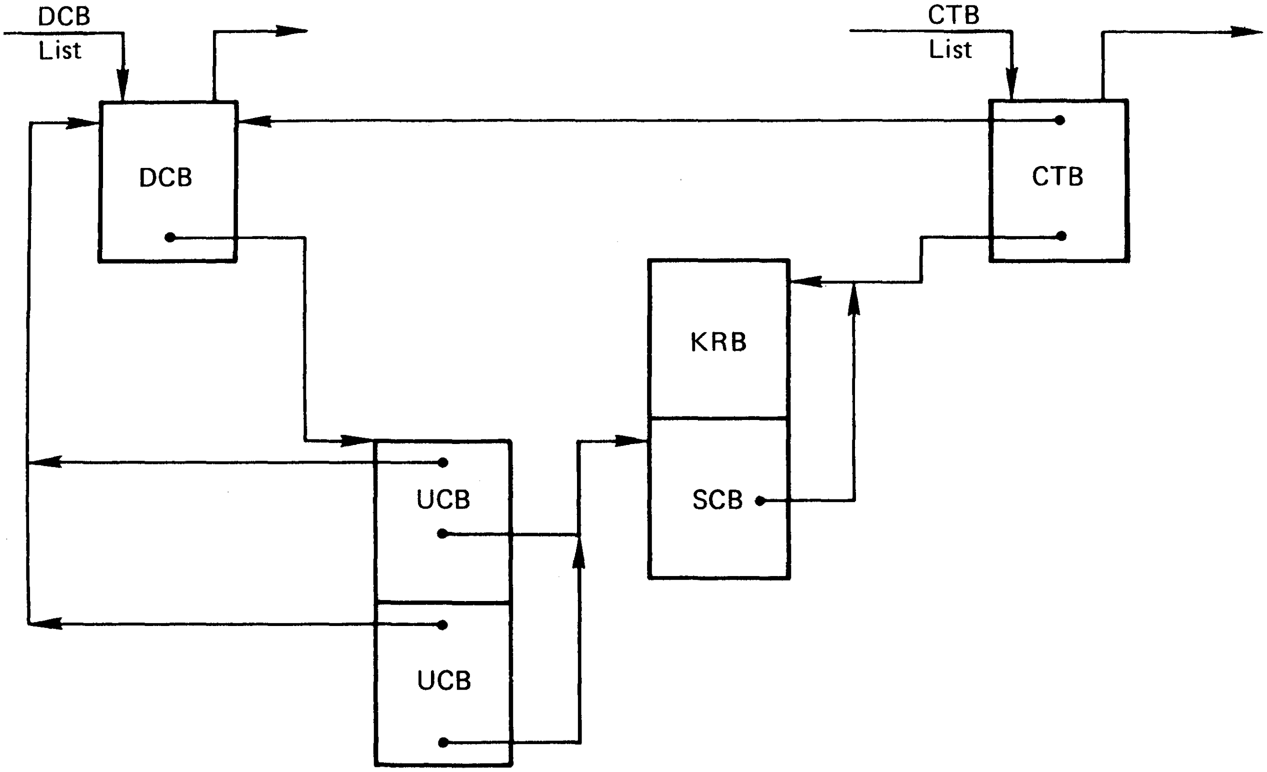
This section presents different arrangements of the control structures that are found in P/OS. The section concentrates on the relationships among device control, unit control, status control, and controller request blocks and controller tables based on hardware and functions supported. Descriptions of the detailed contents of the structures is left to Chapter 4, where the coding requirements are presented. Some of the arrangements are not conventional but are shown to convey the flexibility. Section 2.4 shows how such arrangements fit into the overall system I/O data structure.

The arrangements described in this section illustrate the strategy in offering a flexible I/O data structure. There need be only one controller table for each controller type. Multiple-device control blocks for a single device type reflect the capability to handle varying characteristics. The existence of one or more status control blocks depends on the degree of parallelism possible: one SCB for each controller servicing several units (no parallelism); or one for each device unit combination on the same controller (unit operation in parallel).

The I/O data structure reflects the hardware configuration that the data structures describe. The flexibility in the data structure arrangements provide flexibility in configuring I/O devices. The information density in the structures themselves reduces the coding requirements for the associated drivers.

2.3.1 Multiple Units per Controller, Serial Unit Operation

A typical arrangement of structures for a user-written driver is shown in Figure 2-1. The arrangement could represent an RX50 controller with dual drives. A single controller table (CTB) defines the existence of the controller type on the system. One device control block (DCB) establishes the characteristics for the type of device running on the controller.



ZK-249-81

Figure 2-1: Multiple Units per Controller, Serial Unit Operation

The status control block (SCB) and controller request block (KRB) are contiguous in this arrangement because the software does not allow another I/O operation to begin while the controller is busy. A separate unit control block (UCB) describes each unit attached to the controller. The UCBs are associated with the SCB, which contains the context of the operation currently in progress.

2.3.2 Multiple Controllers, Single Unit per Controller

Another typical conventional arrangement of structures for a user-written driver is shown in Figure 2-2, which could represent two Winchester controllers, one with an RD50 and the other with an RD51 attached. It represents the simplest case of driver processing. Figure 2-1 shows what is required for a controller that allows only a single I/O operation for each controller. A single controller table defines the existence of the controller type on the system. One device control block establishes the characteristics for the type of device running on the controller.

The status control and controller request blocks are contiguous in this arrangement because, while the controller is busy, another I/O operation cannot begin. Only one SCB is necessary to store the context of the unit operation. The UCB points to the SCB, which in turn points to the KRB of the unit's controller. Because the system must handle interrupts from multiple controllers, the controller table points to the KRB of each controller present.

2.3.3 Parallel Unit Operation

Some devices allow multiple units to have operations in progress at the same time. For example, a controller could allow seek operations to overlap data operations. Figure 2-3 shows the arrangement needed in the software structures to support parallel operations on one controller.

Two additional structural changes are required from the serial operation arrangement. First, because more than one unit may have an operation pending at the same time, a structure is needed to store unit context. Therefore, for each unit (and each unit control block) there is a separate status control block. Second, because interrupts can come from more than one unit, some way must exist to access the proper unit. As a result, the controller request block contains a table of unit control block addresses that allows the driver to find the structures for the unit generating an interrupt.

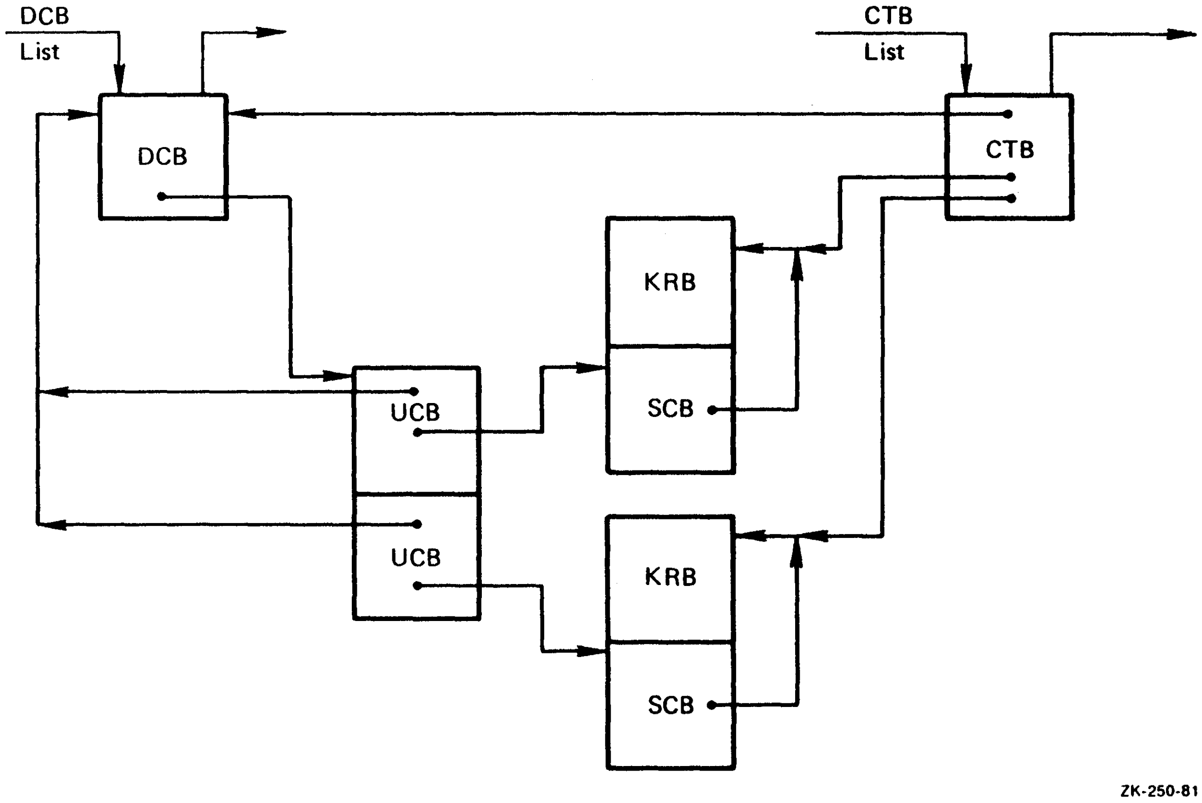


Figure 2-2: Multiple Controllers, Single Unit per Controller

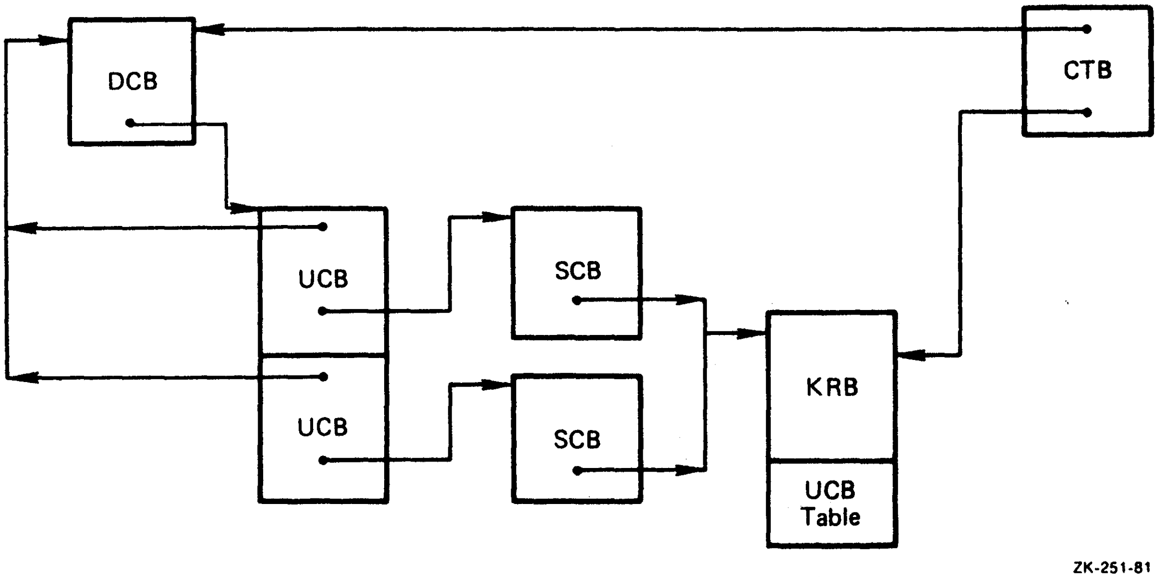


Figure 2-3: Parallel Unit Operation (Overlapped Seek)

2.4 OVERVIEW OF DATA STRUCTURE RELATIONSHIPS

This section presents an overview of the relationships among the user-written driver data structures previously introduced in this chapter and the Executive I/O structures and DIGITAL-supplied driver structures. The goal of the section is to convey the general manner in which user-written structures and code link into the system I/O scheme and to describe generally the use to which the system puts the structures. The specific user-written structures are simplified somewhat so that the emphasis is placed on the linkages with other parts of the system rather than on the details of user-written structural relationships.

This section should be used with Section 2.3 to understand the general structural concepts. For example, Section 2.3 describes various arrangements of unit control, status control, and controller request blocks based on hardware functions the software structures support. This section treats such arrangements as an engineering black box that is oriented in the general I/O environment. Thus, in the generalized I/O data structure depicted in this section, the pointers in the KRB table of the SCB are not shown and the table is simply marked KRB Table.

Figure 2-4, which provides the basis for the presentation of the I/O data structure, shows the individual elements and the important link fields within them. The numbers in the figure correspond to the numbers in the lead paragraphs of the text to simplify the discussion and to guide you through the data structures.

1. The location represented by the Executive symbol $DEVHD is a cell in system common (SYSCM). It is the head (or start) of a singly-linked, unidirectional list of all device control blocks in the system. The first word in each DCB is a link to the next DCB.

   The list of device control blocks is one of the two threads through the system data tables for device-related information. For example, the list is the means by which executive routines scan the data structures to determine what devices are on the system and what is the status of units. User-written device control blocks must be linked into the list of system defined DCBs.

2. Every driver is associated with a partition control block (PCB). The PCB defines the characteristics of the memory area into which the driver is loaded. The Executive and services such as PROLOD reference the data in the PCB. A driver is not concerned with the PCB.

3. If a task is attached to a unit, the UCB has a pointer to the task control block (TCB) of that task.

4. The task header is an independent entity in the I/O data structure and the driver never accesses it. (In fact, it may not be memory resident.) The task header is in physical memory immediately before the task region when the task's "task region" is resident in memory.

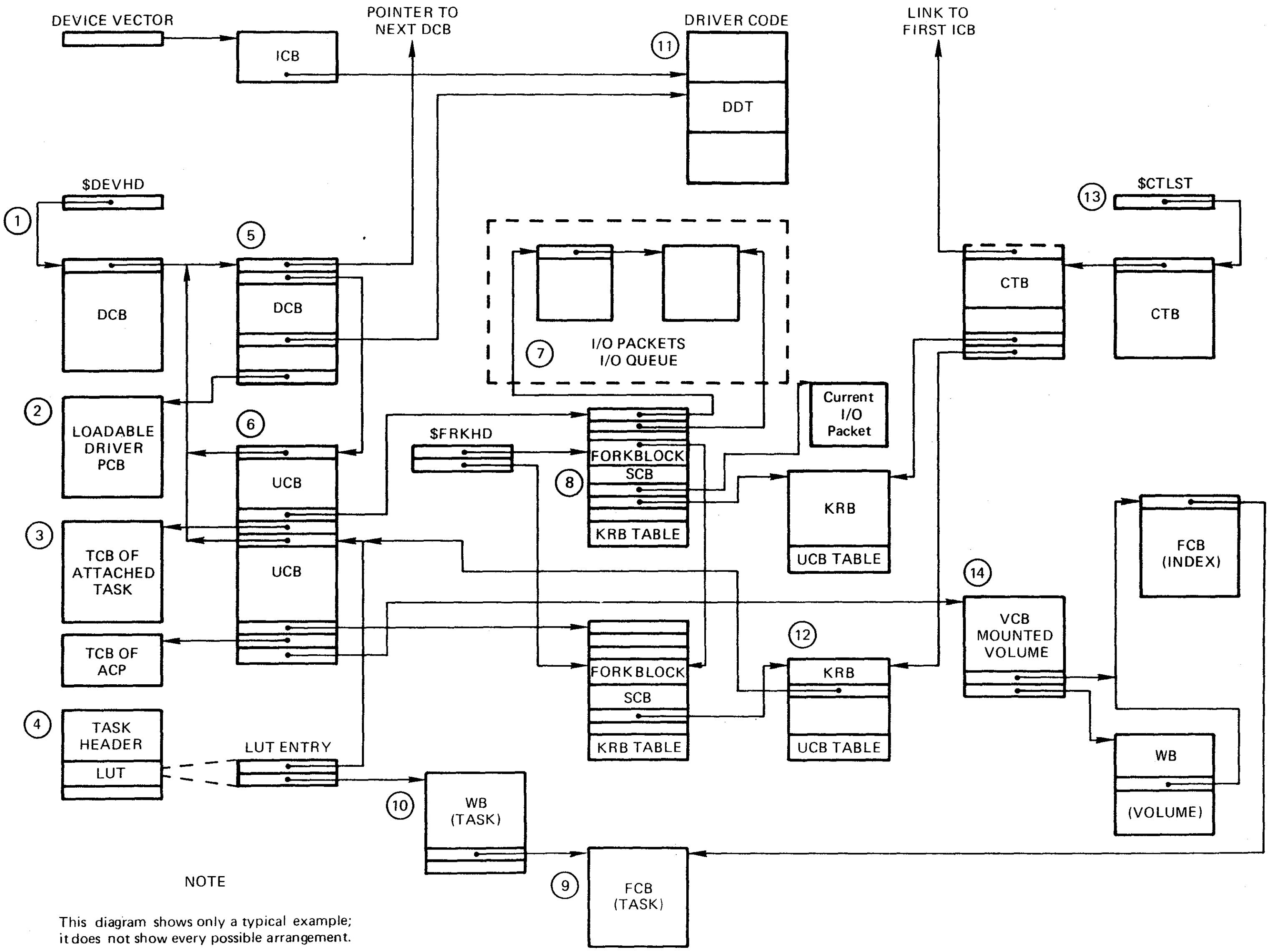
   A logical unit table (LUT) entry in the task header has two items of interest: a pointer to an associated unit control block and, if a file is being accessed, a pointer to a window block. The Executive accesses the logical unit table of a task during a QIO request and indexes the table by the logical unit number specified in the QIO request.

5. A device control block has a pointer to the unit control block of the first related unit. Because the length of a UCB is stored in the DCB and all UCBs are allocated in a contiguous area, access to all the UCBs related to that DCB is possible. This arrangement allows software to access all related unit information for a device type.

   A DCB also has a pointer to the start of the driver dispatch table. This pointer allows the Executive to call the driver at its entry points to process an I/O-related request.

6. Each unit control block contains a pointer back to its related DCB. This backpointer allows the Executive QIO directive to preprocess an I/O request and possibly call the driver (through the pointer to the driver dispatch table).

   Associated with each UCB is a status control block. The SCB is shared by all units for a device type that does not require units to operate in parallel. When units can operate in parallel, each UCB has its own associated SCB.



NOTE

This diagram shows only a typical example; it does not show every possible arrangement.

Figure 2-4: Composite I/O Data Structures

7. As part of processing a QIO directive (queued I/O request), the Executive builds a structure called an I/O packet. Storage for packets is in the system dynamic storage region (the primary pool). The Executive connects the packets by a pointer in each packet to form a linked list called the I/O queue. The Executive maintains two pointers in the SCB to the list of packets. The first pointer is to the start of the list and the second pointer is to the last packet in the list.

   Normally, the driver should not access the list of I/O packets directly. When the Executive transfers control to the driver to initiate processing of an I/O request, the driver immediately calls an Executive service routine to get a packet to process. The routine passes, to the driver, data sufficient to process the request (for example, the address of the packet). Thus, the Executive, and not the driver, removes a packet from the queue of packets. However, in performing the I/O request, the driver can access certain fields in the packet to be processed because a pointer to the currently active I/O packet is kept in the SCB.*

   The Executive determines the ordering of packets in the queue. Typically, higher-priority requests are placed at the head of the queue.

8. At least one status control block (SCB) exists for each controller. Where a controller and software support operations in parallel on multiple units, one SCB exists for each unit capable of operating independently. A pointer in the SCB connects to the controller request block (KRB) of the controller to which the related unit is connected.

   The fork block in the SCB contains some of the driver process context. The driver executes an Executive routine so that processing will occur at fork level. To preserve processing status, the routine stores some context in the fork block. When the driver eventually runs again, the fork processing restores the proper context from the fork block.

   The fork blocks for pending driver processes are connected in a singly-linked list, the head of which is in a location

---

* Normally, the driver does not directly manipulate the I/O queue. An exception is when a driver needs to examine an I/O packet before it is queued or instead of having it queued. This exception involves a status bit in a control byte of the unit control block. For more information on queuing of I/O packets to the driver, refer to the description of the UC.QUE bit in Section 4.4.4.

($FRKHD) in the Executive region. Generally, the fork processing routines link a fork block in FIFO order. At location $FRKHD+2 the executive maintains a pointer to the last fork block in the list.

9. Associated with each open file on a mounted volume is a file control block (FCB). The file system alone uses the FCB to control access to the file.

10. For each open file on a mounted volume, a window block exists for each task that has the file open to hold pointers to areas on the volume on which the file resides. The function of the window block is to speed up the process of retrieving data items from the file. (The associated ACP need not be called to convert a virtual block number in a file to a logical block number on the device.) The driver is not concerned with the window block or this VBN to LBN conversion.

11. The driver dispatch table (DDT) is part of the driver code and is the means by which the executive calls the driver.

12. The controller request blocks (KRB) are linked into the I/O data structure through the pointers in the controller table (CTB).

    The KRB table in the CTB allows the Executive access to the structures for a controller when it initiates an interrupt. To report the termination of a data transfer command, a controller initiates an interrupt. (While such a controller-initiated interrupt is in progress, the hardware delays interrupts from units.) The driver determines the correct KRB by indexing the CTB with the controller index.

    For a controller that allows unit operation in parallel (overlapped seek support), the related KRB must have a table of UCB addresses. This table allows the driver to access the structures of the unit that generates an interrupt. When a unit interrupts, its controller has the physical number of the interrupting unit. The driver must retrieve the number and use it to index the UCB table in the KRB to access the proper unit control block. (For example, see DZDRV.MAC interrupt "B", door open, processing.)

    To support multiple parallel unit operations, the KRB also contains a queue to regulate controller access. This queue, known as the controller request queue, is a list of fork blocks for driver processes that have requested and have been denied immediate access to the controller. If the driver requests access to a controller and the controller is busy, the Executive forces the driver to wait for access by placing the fork block in the queue of processes waiting for access. The Executive gives precedence to control access over requests for data transfer by placing positioning requests onto the front of the queue and adding data transfer requests to the end of the queue. When a unit is given access, the controller status is set to busy and unit UCB address is set to connect the KRB to the owned UCB.

To indicate what unit to process on a controller initiated interrupt, a cell in the KRB points to the unit control block (UCB) of the unit that currently owns the KRB (data transfer).

The KRB controller request queue listhead consists of two words. The first word points to the fork block in the SCB of the next unit to get access. The second word points to the fork block in the SCB of the last unit to get access. If the first word is 0, then the second word points to the first and no unit is waiting for access to the controller.

13. The location represented by the Executive symbol $CTLST is a cell in system common (SYSCM). It is the head (or start) of a singly-linked, unidirectional list of all controller tables (CTBs) in the system. A word in each CTB is a link to the next CTB. The last CTB in the list contains a link word of 0.

    The list of controller tables is one of the two threads through the system for device-related information. (The list of device control blocks is the other thread.) A user-written controller table will be linked into the list of system-defined CTBs. This list is the mechanism by which system routines access I/O data structures for hardware information.

14. One volume control block (VCB) exists for each mounted volume in the system. The VCB maintains volume-dependent control information.

    Pointers within the VCB connect to the file control block (FCB) and window block (WB). The FCB and WB control access to the volume's index file, which is a file of file headers. All FCBs for a volume form a linked list starting from the index file FCB. These linkages aid in keeping file access time to a minimum. A conventional driver never accesses any of these structures.

CHAPTER 3

EXECUTIVE SERVICES AND DRIVER PROCESSING

The Executive provides services related to I/O drivers. Some services are provided before a driver process is initiated and are therefore called predriver initiation services. The predriver initiation services are those performed by the Executive during its processing of a QIO directive; these services are not available as Executive calls.

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

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

An important concept used in this section and in Chapter 4 is the state of a process. In P/OS, a process can run in one of two states, user or system. Drivers operate entirely in the system state; the programming standards described in Chapter 4 apply to system-state processes.

3.1 FLOW OF AN I/O REQUEST

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

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

- The only I/O request in the system is the sample request under discussion.
- The example progresses without encountering any errors that would prematurely terminate its data transfer; thus, no error paths are discussed.
- The controller in question executes only a single operation at a time.

3.1.1 Predriver Initiation Processing

The I/O flow proceeds as described below:

1. Task issues QIO directive

   The user program first either statically (by QIOW$C, QIOW$, QIO$C, or QIO$) or dynamically (by QIOW$S or QIO$S) creates a directive parameter block (DPB) containing information about what I/O is to be performed on what device. Then, it issues the directive.

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

2. QIO Dispatching

   The Executive directive dispatcher DRDSP ascertains that the EMT is a QIO directive and calls the QIO directive processor DRQIO.

3. First-level validity checks

   The QIO directive processor validates the logical unit number (LUN) and the Unit Control Block (UCB) pointer. DRQIO checks whether the LUN supplied in the directive parameter block is a legal value. If it is not a legal value, the directive is rejected. If the LUN is legal, DRQIO checks whether a valid UCB pointer exists in the Logical Unit Table (LUT) for the specified LUN. This check ascertains whether the LUN is assigned. If the check fails, the directive is rejected. If both these checks are successful, DRQIO then performs the redirect algorithm.

4. Redirect algorithm

   Because the UCB may have been dynamically redirected by a Redirect command, QIO directive processing traces the redirect linkage until the target UCB is found. The target UCB provides the links to most of the other structures of the device to which the I/O operation will be directed.

5. Additional validity checks

   The event flag number (EFN) is validated, as well as the address of the I/O Status Block (IOSB). If either is illegal, the directive is rejected. Immediately following successful validation, DRQIO resets the event flag and clears the I/O status block.

6. Obtain storage for and create an I/O Packet

   The QIO directive processor now acquires a 20-word block of dynamic storage for use as an I/O Packet. It inserts into the packet the device-independent data items that are used subsequently by both the Executive and the driver in fulfilling the I/O request. Most items originate in the requesting task's Directive Parameter Block (DPB).

   At this point, DRQIO sets the directive status to +1, which indicates directive acceptance. Note that a directive rejection is a return to the caller with the C bit set. In addition, a directive rejection is transparent to the driver.

7. Validate the function requested

   If the function is legal, DRQIO checks to see whether the unit is on-line. If the unit is off-line, the packet is rejected. The function is one of four possible types:

   Control

   No-op

   ACP

   Transfer (default if not control, no-op, or ACP)

With the exception of Attach/Detach, control functions are queued to the driver. If the bit UC.ATT is set, Attach/Detach will also be queued to the driver. If the requested function does not require a call to the driver, the Executive takes the appropriate action and calls the I/O Finish routine ($IOFIN).

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

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

Transfer functions are address checked and queued to the proper driver. This means that DRQIO checks the address of the I/O buffer, the byte count, and the alignment requirement for the specified device. If any of these checks fails, DRQIO calls the I/O Finish routine ($IOFIN), which returns an I/O error status and clears the I/O request from the system. If the checks succeed, DRQIO either places the I/O Packet in the driver request queue according to the priority of the requesting task or, if the UC.QUE bit is set, gives the packet directly to the driver. (See Section 4.4.5 for a description of the UC.QUE bit.)

3.1.2 Driver Processing

1. Request work

   The Executive passes control to the driver by using the driver's initiation entry point for each new I/O request.

   To obtain work, the driver calls Get Packet ($GTPKT). $GTPKT either provides work, if it exists, or informs the driver that no work is available or that the SCB is busy; if no work exists, the driver returns to its caller. If work is available, $GTPKT sets the device controller and unit to busy, dequeues an I/O request packet, and returns to the driver.

2. 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.

3. Interrupt processing

   When a previously issued I/O operation interrupts, the interrupt causes the driver to be entered. The driver processes the interrupt according to the programming protocol described in Chapter 1. According to the protocol, the driver may process the interrupt at priority 7, at the priority of the interrupting device, or at fork level. If the processing of the I/O request associated with the interrupt is still incomplete, the driver initiates further I/O on the device (Step 9). When the processing of an I/O request is complete, the driver calls $IODON.

4. I/O Done processing

   $IODON removes the busy status from the device unit and controller, queues an AST if required, and determines whether 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 initiation entry point and looks for more work (Step 1). This procedure is followed until the driver cannot obtain work; then the driver returns to its caller.

   Eventually, the processor receives a ready-to-run task that issues a QIO directive, starting the I/O flow all over again.

3.2 EXECUTIVE SERVICES AVAILABLE TO A DRIVER

Once a driver is given control following an I/O interrupt or by the Executive itself, a number of Executive services are available to the driver. These services are discussed in detail in Chapter 7.

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

1. Get Packet ($GTPKT)

2. Create Fork Process ($FORK)

3. I/O Done ($IODON or $IOALT)

3.2.1 Get Packet ($GTPKT)

The Executive, after it queues an I/O Packet, calls the appropriate driver at its I/O initiation entry point. The driver then immediately calls the Executive routine $GTPKT to obtain work.* If work is available, $GTPKT delivers to the driver the highest-priority, executable I/O Packet in the driver's I/O queue, and sets the SCB status to busy. If the driver's I/O queue is empty or if the driver is busy, $GTPKT returns a no-work indication.

If the SCB related to the device is already busy, $GTPKT so informs the driver, and the driver immediately returns control to the Executive.

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

3.2.2 Create Fork Process ($FORK)

Synchronization of access to shared data bases is accomplished by creating a fork process. When a driver needs to access a shared data base, it must do so as a fork process; the driver becomes a fork process by calling $FORK. The SCB contains preallocated storage for a 5-word fork block. See Section 4.4.5 for a description of the fork block. Section 1.3.3 contains details on $FORK. After $FORK is called, a routine is fully interruptable (priority 0), and its access to shared system data bases is strictly linear.

3.2.3 I/O Done ($IODON or $IOALT)

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

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

$IODON and $IOALT also declare a significant event, reset the SCB and device unit status to idle, and release the dynamic storage used by

---

* An exception is a driver that handles special user buffers. Such a driver must call certain other Executive routines before calling $GTPKT. See Section 4.4.4 for a description of the UC.QUE bit.

the completed I/O operation.

CHAPTER 4

PROGRAMMING SPECIFICS FOR WRITING AN I/O DRIVER

Chapters 2 and 3 give overviews of data structures and Executive services, respectively. This chapter summarizes programming standards, presents overviews of programming requirements for user-written driver code and data, and gives details of the data structures and driver code. Executive services are covered in Chapter 7.

4.1 PROGRAMMING STANDARDS

I/O drivers function as integral components of the P/OS Executive, and this manual enables you to incorporate I/O drivers into your system. User-written drivers must follow the same conventions and protocol as the Executive itself if they are to avoid complete disruption of system service. Failure to observe the internal conventions and protocol that are described fully in Chapter 1 can result in poor service and reductions in system efficiency.

The programming conventions used by P/OS system components are identical to those described in Appendix E of the *PDP-11 MACRO-11 Language Reference Manual.* DIGITAL urges you to adhere to these conventions.

4.1.1 Programming Protocol Summary

Drivers are required to adhere to the following internal conventions when processing device interrupts:

1. Registers R4 and R5 are available; any other registers must be saved and restored.

2. Processing at the priority of the interrupting source should be minimized and kept well under 500 usecs. On Professional Series hardware, all devices interrupt at processor priority 4 and, as a result, processing at device priority is equivalent to processing at priority 7 with all pending interrupts (including the clock) locked out. The interrupt arbitration described in the "Professional 300 Series Technical Manual" only addresses which of the interrupts is subsequently serviced at priority 4 when the processor drops its priority to 0.

3. Kernel APR 6 mapping must be preserved by the interrupt service routine if needed to map additional driver code or data buffers.

4. Only a fork process, which by definition is in system state, may modify a system data base or examine dynamic system data structures such as the installed task directory (the STD).

5. A fork process has unrestricted access to APR 6, and R0-R5.

6. Complex drivers and system processes that require extended periods of processing at system state should consider reforking to allow other system processes' execution time. These other system processes could, for example, perform clock-related and I/O completion-related processing. Care must be exercised that any additional context is preserved if required, since only R4, R5 and APR 5 mapping are preserved across the fork. As always, "double forking" must not occur; it is avoided by either using an interlock protocol on the forkblock, or through the use of a separate forkblock allocated from primary pool.

4.1.2 Accessing Driver Data Structures

All the driver data structure elements have symbolic offsets. Because the physical offset values may vary from one version of the Executive to another, your user-written driver code should always use the symbols to access the elements.

Accordingly, your driver code should not step from one structural element to another (relying on the juxtaposition of data structures and individual words in a data structure) but should access each element by symbolic offset. On the other hand, it is a common coding practice to assume that zero offsets (particularly link pointers such as D.LNK) will remain zero. This assumption allows the saving of one word per instruction by substituting an instruction such as MOV (R3),R3 for MOV D.LNK(R3),R3. DIGITAL recognizes that such practices are followed and consequently attempts to keep such offsets zero.

4.2 OVERVIEW OF PROGRAMMING USER-WRITTEN DRIVER DATA BASES

You should create the source code for your user-written driver data base in a file separate from that of the driver code. You assemble this file to create the driver data base module. If your data base is in a separate module, it will be linked to the end of the driver code module. If your driver data base is in the same module as that of your driver code, it must be at the end of the driver code.

To create the source code, you need to know, in addition to the detailed structures, what ordering and labeling are required. These requirements, though not extensive, are important in linking and loading your driver data base. The general coding requirements for driver data bases are described in the following subsections.

4.2.1 General Labeling and Ordering of Data Structures

When creating a data base, you must specify, for the PROLOD routines, two global labels as follows:

$xxDAT::	marks the start of the user-written driver data base.
$xxEND::	marks the end of the user-written driver data base, that is, immediately following the final word of the data base.

If either or both of these labels are not defined, PROLOD cannot determine the length of your data base when you attempt to load your driver.

There is no mandatory ordering of the different structures in a driver data base. DIGITAL suggests, however, that you place the DCB first, followed by the UCB, the SCB(s), the KRB(s), and the CTB. If you do not follow this ordering scheme, you must specify the starting location of the first (or only) DCB as described in Section 4.2.2.

4.2.2 Device Control Block Labeling

When writing a driver data base, the PROLOD routines require either that the first (or only) DCB be identified by the global label $xxDCB:: or that the DCB be at the start of the data base.

4.2.3 Unit Control Block Ordering

All the UCBs associated with a specific device control block (DCB) must be contiguous with each other and must be of equal length. These requirements are necessary because the DCB has only one link to the UCBs, and that link is to the first UCB. Two data elements, the UCB length and the number of units, are stored in the DCB; they, together with the link to the first UCB, are used to locate subsequent UCBs. If you do not follow these requirements, no software can access the UCBs.

4.2.4 Status Control and Controller Request Blocks

All user-written drivers that do not need separate storage for independent unit context should use the contiguous allocation of the KRB and SCB. (For an explanation of when independent unit context is required, refer to the discussion of overlapped seek I/O in Section 1.4.1. Therefore, the KRB and SCB are contiguous and some fields of each structure overlap. This arrangement saves space that would be required for one SCB for each independent unit. Because only one unit can be active at any one time, all units attached to the same controller can share the SCB. This arrangement of the KRB and the SCB is described in Section 4.4.7.

4.2.5 Controller Table

You must define the start of the table of KRB addresses in the CTB with the global label $xxCTB::. Both the INTSV$ macro call and the Executive PROLOD routines require this label. You must assemble a global symbol of the form $xxCTB in the CTB starting at the first word in the table of KRB addresses. An example follows.

```
                .
                .
                .
                 CTB storage cells ...
                .
                .
                .
                .WORD $XXDCB    ;Offset L.DCB in CTB
                .BYTE   3       ;L.NUM, where 3 is # of KRBs (controllers
                .BYTE   0       ;L.STS, controller status
        $XXCTB::                ;Start of table of KRB addresses
                .WORD   KRB1    ;Address of 1st KRB
                .WORD   KRB2    ;
                .WORD   KRB3    ;
```

The symbol $xxCTB (defined globally, where xx is the two character driver generic name) is used by PROLOD to find the CTB in your driver data base image. The value of this symbol (that is, the address) is placed into the DDT in the word labeled by xxCTB (no $). The INTSV$ macro references the value of the word in the DDT labeled by xxCTB to obtain the address (in the CTB) of the table of KRB addresses at runtime.

4.3 OVERVIEW OF PROGRAMMING USER-WRITTEN DRIVER CODE

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

1. Thoroughly read and understand this manual.
2. Familiarize yourself in detail with the physical device and its operational characteristics.
3. Determine the level of support required for the device.
4. Determine actions to be taken at the driver entry points.
5. Create the driver source code.

To assist you in generating proper code for your user-written driver and to provide a stable user-level interface from one release of the system to another, P/OS provides the macro calls listed in Table 4-1.

The definitions of the system macro calls for drivers are in the Executive assembly prefix file RSXMC.MAC. The following subsections describe the format of the macro calls and other features of user-written driver code. Driver code details (such as labeling requirements and entry point conditions) are presented in Section 4.5.

4.3.1 Generate Driver Dispatch Table Macro Call - DDT$

The DDT$ macro call facilitates generation of the driver dispatch table. The format of the DDT$ macro call is as follows:

```
DDT$    dev,nctrlr,iny,inx,ucbsv,new,buf,opt
```

Table 4-2 lists the arguments of the DDT$ macro call. The macro constructs the DDT, using as addresses those locations indicated by the standard labels. The macro has arguments allowing you to tailor some of the standard entry points. The format of the DDT generated by the DDT$ macro is described in Section 4.5.1.

---

* See Appendix A for Macro definitions.

Table 4-1: System Macro Calls for Driver Code*

Macro Name	General Functions
DDT$	Used conventionally at the start of the driver code (1) to allocate storage for and to generate a driver dispatch table containing the addresses of entry points in the order in which the Executive expects them; (2) to generate special global labels required by the Executive; (3) to tell the Executive PROLOD routines: (a) which controllers the driver supports, (b) how many interrupt vectors each controller supports, and (c) the association between the interrupt vectors and the driver interrupt entry points; and (4) to generate default controller and unit status change entry point procedures (for on-line and off-line transitions)
GTPKT$	Used at the I/O initiator entry point to generate the call to the $GTPKT routine and to generate code to save the address of the currently active unit's UCB
INTSV$	Used at an interrupt entry point to conditionally generate a call to the $INTSV routine and to generate code to load the UCB address of the interrupting device into R5

Table 4-2: DDT$ Macro Call Arguments

Argument	Meaning
dev	Is the 2-character device mnemonic (used to generate entry point symbol names such as a $xxINI where dev=xx).
nctrlr	Is the number of controllers that the driver services (counting from 1).
iny	Allows the definition of no interrupt entry point or multiple interrupt entry points. If you leave the argument null, the macro generates as the interrupt

	entry point address the location defined by the conventional label $xxINT. If you specify NONE, no interrupt entry point is generated for the controller. If you specify an argument list of the form <aaa,bbb>, the macro generates multiple cells containing addresses defined by unconventional labels of the form $xxaaa and $xxbbb. This latter mechanism allows you to define multiple interrupt entry points in the driver. For example, the argument list <INP,OUT> generates two interrupt address labels of the form $xxINP and $xxOUT, the typical names used by drivers with two interrupt entry points.
inx	Uses an alternate I/O initiation entry point address label instead of the conventional INI form. If you specify inx, the macro uses as the only I/O initiation entry point address the location defined by the label inx.
ucbsv	This argument is optional on P/OS systems. Section 4.3.4 provides guidelines on specifying this argument. If this argument is non-blank, a table of nctrlr words in length is generated to contain the controller index to UCB mapping of a per controller I/O operation in progress.
new	If present, causes the DDT$ macro to generate the DDT table that refers to the online reconfiguration entry points. (See the macro definition of DDT$ in Appendix A.)
buf	Required if the driver performs buffered input and output. The entry point xxDEA: is generated.
opt	Required if the mass storage device driver supports queue optimization.

4.3.2 Get Packet Macro Call - GTPKT$

The GTPKT$ macro call standardizes use of the Executive $GTPKT routine, which retrieves an I/O packet for the driver to process. The format of the GTPKT$ macro call is as follows:

```
     GTPKT$    dev,nctrlr,addr,ucbsv,suc
```

The description of the arguments appears in Table 4-3.

Table 4-3: GTPKT$ Macro Call Arguments

Argument	Meaning
dev	Is the 2-character device mnemonic.
nctrlr	Is the number of controllers that the driver services (counting from 1).
addr	Is the local label defining the location at which to continue execution if there is no I/O packet available. A driver typically executes a RETURN instruction when the $GTPKT routine indicates that there is no I/O packet to process. If you leave this argument null, therefore, the macro generates a RETURN instruction.
ucbsv	Strictly speaking, this argument is not needed on P/OS systems and can not be used directly if a given controller can support parallel operations on multiple units simultaneously. If this argument is non-blank, a table of "nctrlr" words in length is generated to contain the controller index to UCB mapping of a per controller I/O operation in progress. The macro then generates code to load the pointer S.OWN with the address of the UCB returned by $GTPKT. For guidelines on using the argument, refer to Section 4.3.4.
suc	Indicates single unit controller. If you are writing a driver that supports a controller type such as the LP11, to which only a single unit can be attached, you should specify this argument (any character(s) except null). If you specify this argument, you should ensure that the offset K.OWN/S.OWN in the KRB(s) of your driver data base points to the UCB(s) of the unit(s) to which the controller(s) is attached. Thus, the macro does not generate code that stores the UCB

address in the KRB for a device that has only one UCB per KRB.

If your driver has multiple units attached to the same controller, you should leave this argument null. The macro will then generate code to store the UCB address of the unit to process in the SCB or SCB/KRB.

Note that for non-contiguous SCB/KRB configurations, the UCB is stored in K.OWN when the controller request is granted, depending on controller characteristics (see $RQCNC and $RQCND in MDSUB).

This macro call generates the call to the Executive $GTPKT routine. You should place it at the I/O initiation (xxINI) entry point because the $GTPKT routine is the standard manner for a driver to receive work from the Executive. When the driver receives control at its INI entry point, the Executive has loaded R5 with the address of the UCB of the unit that the driver must service. Because of the code the macro call generates, the driver immediately calls $GTPKT, which can set the C bit to indicate that no work is pending. The call additionally generates the BCS instruction that returns control to the calling routine when there is no work. If you specify an address as the "addr" argument in the macro call, it is used as the destination of the BCS instruction. The address is typically that of a RETURN instruction, but does not have to be. Eventually the driver must execute a RETURN to the system.

The $GTPKT routine indicates that the driver has an I/O packet to process by clearing the C bit. Therefore, when the test of the BCS instruction is false, execution continues inline and the driver can process the I/O packet that the Executive queued to it. The $GTPKT routine leaves information in the driver registers to enable the driver to process the request. Refer to the description of the $GTPKT routine and the GTPKT$ macro listing in Chapter 7.

4.3.3 Interrupt Save Macro Call - INTSV$

You should specify the INTSV$ macro call at each interrupt entry point in the driver. The format of the INTSV$ macro call is as follows:

```
INTSV$    dev,pri,nctrlr,pswsv,ucbsv
```

The arguments of the call are described in Table 4-4. The macro generates the code to load R5 with the UCB address of the current controller owner, given the controller index is in R4 (R4 is not modified by macro). Note that R5 may be zero in case of either an unexpected interrupt (e.g., no current I/O operation), or an interrupt as a result of a parallel operation on the same controller. (For example, an overlapped seek completing after a data transfer completion.) In general, for configurations which support overlap seek it is the driver's responsibility to differentiate between a transfer function completion interrupt from a control function completion interrupt.

Table 4-4: INTSV$ Macro Call Arguments

Argument	Meaning
dev	Is the 2-character device mnemonic.
pri	Is not used.
nctrlr	Is the number of controllers that the driver services (counting from 1).
pswsw	Leave this argument null; it has no effect. It is an anachronism, and must be positionally present if ucbsv is specified.
ucbsv	Strictly speaking, this argument is not needed on P/OS systems and can not be used directly if a given controller can support parallel operations on multiple units simultaneously. If this argument is non-blank, a table of "nctrlr" words in length is generated to contain the controller index to UCB mapping of a per controller I/O operation in progress. The macro generates code which uses the controller index returned in R4 by $INTSI to index into a UCB table to load the UCB address of the interupting device into R5.

4.3.4 Using the UCBSV Argument in Macro Calls

You can optionally specify the ucbsv argument in calls to the macros DDT$, GTPKT$, and INTSV$. This argument allows you to use an alternative technique to map the controller index to a UCB address. This provides a simple mechanism for retrieving the UCB when an expected interrupt occurs.

The argument ucbsv in the DDT$ macro allocates nctrlr words of storage (one word for each controller that the driver supports) and labels the first word ucbsv:. This table contains the address of the unit control block of the interrupting devices for each controller. The GTPKT$ macro updates table entries at I/O initiation and INTSV$ references this table to retrieve the UCB address.

If you specify the argument ucbsv in the GTPKT$ macro call, it must be the same label you supplied for the ucbsv argument in the DDT$ and INTSV$ macro calls. The macro generates code to move the UCB address returned by $GTPKT to the correct location in the table starting at the label ucbsv.

If you specify the argument ucbsv in the INTSV$ macro call, it should be the same label you supplied for the ucbsv argument in the DDT$ and GTPKT$ macro calls. The macro uses ucbsv to locate the UCB address of the interrupting unit, and then generates code to load the address into R5.

4.3.5 Driver Entry Points for PROLOD

A driver that requires additional initialization and completion functions can define two entry points by labels of the form $xxLOA and $xxUNL. Because these two labels do not appear in the DDT itself, their format is fixed; you must use the exact format in your driver code. When you load the driver, the

NOTE

> Drivers should not attempt to access controller registers at the load entry point. Device access is only possible after the driver has been called at its controller online entry point.

PROLOD routines check for the $xxLOA entry point.

The driver is entered, once per UCB, at the $xxLOA entry point at priority zero. At this stage, the driver data base has been loaded and pointers have been relocated. The driver is mapped through APR 5, and the following registers are set up:

R3 Controller index (undefined if S.KRB = 0)
R4 - Address of the status control block
R5 - Address of the unit control block

NOTE

> The driver cannot access any device CSRs at this point because ownership has not been established. To access device CSRs, wait until the controller is online.

The driver may use all the registers. When you unload the driver, PROLOD calls it at the $xxUNL entry point with the same conditions. These two entry points in the driver are independent of the controller and unit status change entry points used by the Executive. That is, the two entry points $xxLOA and $xxUNL are used for initialization and at driver load and unload time and not at on-line and off-line status change time. Note that $xxUNL is called only when all controllers and units are offline. The data base is not removed, but it is reused on subsequent reloads.

4.4 DRIVER DATA STRUCTURE DETAILS

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

1. The I/O packet
2. The DCB
3. The UCB
4. The SCB
5. The KRB
6. The CTB

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

NOTE

> Except where explicitly noted otherwise, all unused bits, fields, and words in all driver data base structures are reserved for DIGITAL system use and expansion.

In the following descriptions, most data fields (words or bytes) are classified by one of five descriptions. Two items in each description indicate:

- Whether the field is initialized in the data-structure source, and

- What sort of access the driver has to the field during execution

The five descriptions are:

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

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

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

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

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

These five descriptions cover most of the fields in the control blocks described in this section. No system software or hardware checks or enforces any of the access described. Exceptions are noted in the text.

4.4.1 The I/O Packet

Figure 4-1 shows the layout of a control function I/O Packet, and Figure 4-2 shows the layout of a transfer function I/O Packet. Both are constructed and placed in the driver I/O queue by QIO directive processing, and subsequently delivered to the driver by a call to $GTPKT. The DPB from which the I/O Packet is generated is illustrated in Section 4.4.2. QIO directive processing dynamically builds the I/O packet from the data in the DPB. Fields in the I/O Packet (see the following text) are classified as:

- Not referenced,
- Read-only, or
- Read-write.

I.LNK

Driver access:

Not referenced.

Description:

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

I.EFN

Driver access:

Not referenced.

Description:

Contains the event flag number as copied by QIO directive processing from the requester's DPB. Bit 0<200> indicates a virtual function.

I.PRI

Driver access:

Not referenced.

Description:

Priority copied from the TCB of the requesting task.

I.LNK	Link to next I/O packet		0
I.PRI I.EFN	EFN	PRI	2
I.TCB	TCB address of requester		4
I.LN2	Address of second LUT word		6
I.UCB	Address of redirect UCB		10
I.FCN	Function code	Modifier	12
I.IOSB	Virtual address of I/O status block		14
	Relocation bias of IOSB		16
	Real address of IOSB		20
I.AST	Virtual address of AST service routine		22
I.PRM	Device parameters (control functions)	P1	24
		P2	
		P3	
		P4	
		P5	
		P6	
I.AADA	Attachment Descriptor Pointer		
I.AADA+2	Attachment Descriptor Pointer		

Figure 4-1: I/O Packet Format - Control Function

I.LNK	Link to next I/O packet		0
I.PRI I.EFN	EFN	PRI	2
I.TCB	TCB address of requester		4
I.LN2	Address of second LUT word		6
I.UCB	Address of redirect UCB		10
I.FCN	Function code	Modifier	12
I.IOSB	Virtual address of I/O status block		14
	Relocation bias of IOSB		16
	Real address of IOSB		20
I.AST	Virtual address of AST service routine		22
I.PRM	Relocation BIAS of buffer		24
	Displacement of buffer (+140000)		
	P2	Device parameters (transfer functions)	
	P3		
	P4		
	P5		
	P6		
I.AADA	Attachment Descriptor Pointer		
I.AADA+2	Attachment Descriptor Pointer		

P1 assumed to be buffer virtual address
P2 assumed to be buffer size in bytes

Figure 4-2: I/O Packet Format - Transfer Function

DRIVER DATA STRUCTURE DETAILS

I.TCB

Driver access:

Not referenced usually. Referenced at I/O cancel.

Description:

TCB address of the requesting task.

I.LN2

Driver access:

Not referenced.

Description:

Contains the address of the second word of the LUT entry in the task header to which the I/O request is directed, if even. Also, for virtual functions, if even, the task region, and consequently the header, was locked in memory by incrementing the appropriate I/O counts. For open files on file-structured devices, this word contains the address of the Window Block. If odd, it is the window block pointer; otherwise zero.

I.UCB

Driver access:

Not referenced by conventional driver; frequently referenced by drivers that use $GSPKT and maintain parallel active unit context UCBs rather that the SCBs, and therefore have a "many to one" UCB to SCB configuration.

Description:

Contains the address of the unit to which I/O is to be directed. I.UCB is the address of the Redirect UCB if the starting UCB has been subject to a Redirect command. The field is referenced by the $GTPKT routine.

I.FCN

Driver access:

Read-only.

Description:

Contains the function code for the I/O request. It consists of two bytes. The high-order byte contains the function code; the low-order byte contains modifier (subfunction) bits. During predriver initiation the Executive compares the function code with a function mask value in the DCB. The driver interprets the modifier (subfunction) bits.

I.IOSB

Driver access:

Not referenced. Do not touch. Driver specifies status at I/O completion in registers to $IODON or $IOFIN. The region containing the IOSB is not guaranteed to be memory resident, except at predriver initiation time (see UC.QUE).

Description:

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

I.IOSB+2 and I.IOSB+4 contain the address doubleword for the IOSB if I.IOSB is even and nonzero (see Section 7.4 for a detailed description of the address doubleword.

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:

Read-write.

Description:

Device-dependent parameters constructed from the last six words of the DPB. Note that if the I/O function is a transfer (refer to the description of D.MSK in Section 4.4.3, the buffer address (first DPB device-dependent parameter) is translated to an equivalent address doubleword. Therefore, the virtual buffer address, which occupied one word in the DPB, occupies two words in I.PRM. As a result, all other parameters in I.PRM are shifted by one word so that device-dependent parameter n is copied to I.PRM +(2*n)+2.

Most DIGITAL-supplied drivers treat these words as a read/write storage area after their initial contents have been used.

When the last word of the device-dependent parameters is nonzero, the value can have one of several special meanings to the Executive. For example, if the value is nonzero and the I/O function is marked "virtual," the Executive assumes that the value is a block locking word. Therefore, if the driver uses the word, it should restore its contents before calling $IODON.

I.AADA
I.AADA+2

Driver access:

Not referenced; maintained by the Executive transparently to the driver.

Description:

Two pointers, each to an attachment descriptor block of the region in which the task I/O buffer resides. These pointers account for I/O by region and enable the Executive to lock a region to make it noncheckpointable while I/O is in progress, and to unlock a region after I/O completes.

4.4.2 The QIO Directive Parameter Block (DPB)

The QIO DPB is constructed as shown in Figure 4-3. Usually drivers never access the DPB; the information is supplied here for general reference.
The parameters in the DPB have the following meanings:

Length (required):

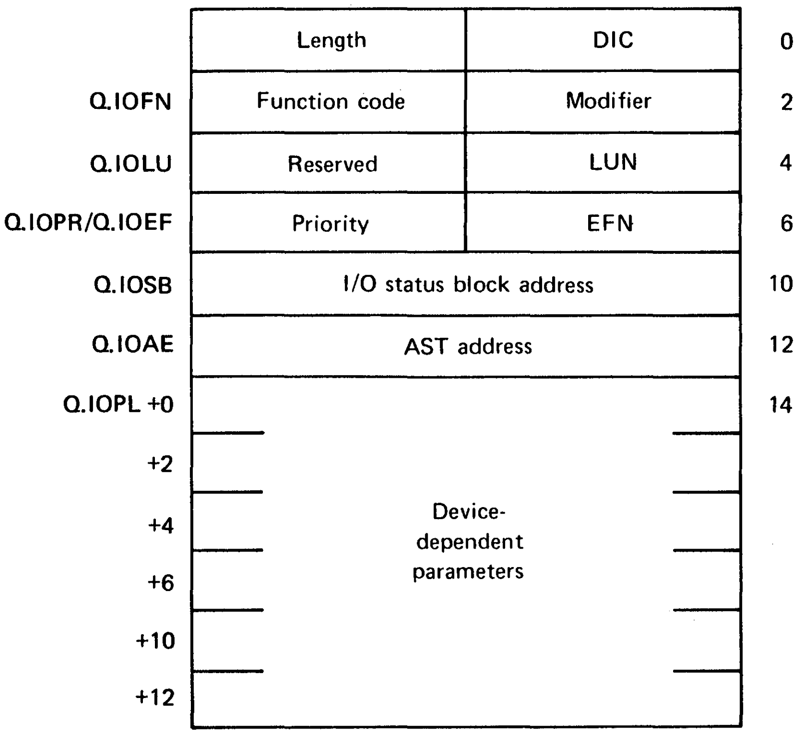
The length of the DPB, which for the QIO directive is always fixed at 12 words.

DIC (required):

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

Q.IOFN (required):

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



ZK-255-81

Figure 4-3: QIO Directive Parameter Block (DPB)

Modifier:

Device-dependent modifier bits.

Reserved:

Reserved byte; must not be used.

Q.IOLU (required):

Logical Unit Number.

Q.IOPR:

Request priority. Ignored by P/OS but space must be allocated for IAS compatibility.

Q.IOEF (optional):

Event flag number. Zero indicates no event flag.

Q.IOSB (optional):

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

Byte 0

I/O status byte.

Byte 1

Augmented data supplied by the driver.

Bytes 2 and 3

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

Q.IOAE (optional):

Address of the I/O done AST service routine.

Q.IOPL

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

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

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

4.4.3 The Device Control Block (DCB)

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

ZK-256-81

Figure 4-4: Device Control Block

The fields* in the DCB are described as follows:

D.LNK (link to next DCB)

---

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

Driver access:

Initialized, not referenced.

Description:

Address link to the next DCB. If this cell is in the last (or only) DCB, you should set its value to zero. If you are incorporating more than one user-written driver at one time, then this field should point to another DCB in a DCB chain, which is terminated by a value of zero.

D.UCB (pointer to first UCB)

Driver access:

Initialized, not referenced.

Description:

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

D.NAM (ASCII device name)

Driver access:

Initialized, not referenced.

Description:

Generic logical 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. The low-order byte contains the lowest logical unit number; the high-order byte contains the highest logical unit number. This range covers those logical units available to the user for device assignment. Typically, the lowest number is zero or one, and the highest is n-1, where n is the number of device-units described by the DCB.

D.UCBL (UCB length)

Driver access:

Initialized, not referenced.

Description:

The unit control block can have any length to meet the needs of the driver for variable storage. However, all UCBs for a given DCB must have the same length. The specified length must include prefix words (such as U.LUIC and U.OWN), if present.

D.DSP (driver dispatch table pointer)

Driver access:

Not referenced.

Description:

Address of the driver dispatch table, which is located within the driver code. (When the Executive wishes to enter the driver at any of the 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.)

D.MSK (driver-specific function masks)

Driver access:

Initialized, not referenced.

Description:

Eight words, beginning at D.MSK, are critical to the proper functioning of a device driver. The Executive uses these words to validate and dispatch the I/O request specified by a QIO directive. The following description applies only to nonfile-structured devices.* Four masks, with two words per

---

* Although no DIGITAL publication describes writing drivers for file-structured devices (drivers that interface with F11ACP), you could write a disk driver by using a DIGITAL-supplied driver as a

mask, are described by the bit configurations that you establish for these words:

1. Legal function mask
2. Control function mask
3. No-op function mask
4. ACP function mask

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

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

The masks are laid out in memory in two 4-word groups. Each 4-word group covers 16 function codes. The first 4 words cover the function codes 0 through 15; the second 4 words cover codes 16 through 31. Below is the exact layout used for the driver example in Chapter 8.

```
.WORD   177477      ;LEGAL FUNCTION MASK CODES 0-15.
.WORD   70          ;CONTROL FUNCTION MASK CODES 0-15.
.WORD   0           ;NO-OP FUNCTION MASK CODES 0-15.
.WORD   177200      ;ACP FUNCTION MASK CODES 0-15.
.WORD   377         ;LEGAL FUNCTION MASK CODES 16.-31.
.WORD   0           ;CONTROL FUNCTION MASK CODES 16.-31.
.WORD   0           ;NO-OP FUNCTION MASK CODES 16.-31.
.WORD   377         ;ACP FUNCTION MASK CODES 16.-31.
```

The Executive filters the function code through the mask words sequentially as follows:

Legal Function Mask:

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

Control Function Mask:

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

No-op Function Mask:

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

ACP Function Mask:

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

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

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

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

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

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

Transfer Function Processing:

Finally, if the function is not an ACP function, then it is by default a transfer function. All transfer functions cause the QIO directive processor to check the specified buffer for legality (that is, inclusion within the address space of the requesting task) and proper alignment (word or byte). In addition, the processor checks the number of bytes being transferred for proper modulus (that is, nonzero and a proper multiple). All transfer functions except IO.WLB and IO.WVB are assumed to require Read and Write access to the buffer region, and are access checked accordingly. By convention, the first user-supplied parameter is the buffer address and the second is the byte count.

Creating Mask Words:

Creating function mask words involves the following five steps:

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

2. Build the Legal Function mask: Check the standard P/OS function mask values in Table 4-6 for equivalencies. Only the IO.KIL function is mandatory. IO.ATT and IO.DET functions, if used, must have the P/OS system interpretation. DIGITAL suggests that functions having an P/OS system counterpart use the P/OS code, but this is required only when the device is to be used in conjunction with an ACP. From the supported function list in Table 4-5, you can build the two Legal Function mask words.

3. Build the Control Function mask by asking:

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

If it does not, then either it qualifies as a control function or the driver itself must effect the checking and conversion of any addresses to the format required by the driver. See Section 8.1 for an example of a driver that does this. (Buffer addresses in standard format are automatically converted to Address Doubleword format.)

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

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

5. Finally, include the ACP functions Write Virtual Block and Read Virtual Block for those drivers that support both read and write. (Include only one related ACP function if the driver supports only read or write). Other ACP functions that might be included fall into the nonconventional driver classification and are beyond the scope of this document.

D.PCB (0)

Driver access:

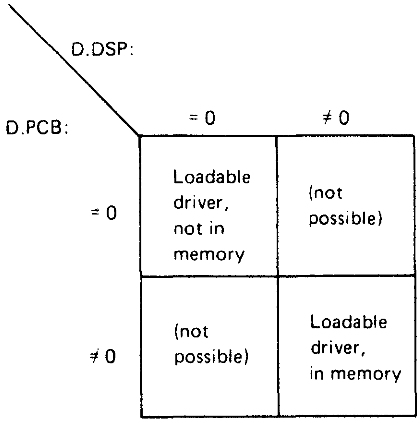
Initialized, not referenced.

Description:

Address of the driver's Partition Control Block (PCB). The driver data base source must initialize the address to zero. The DCB can be extended by adding words after D.PCB. A PCB exists for every partition in a system. A driver PCB describes the partition in which it resides.

The Executive uses D.PCB together with D.DSP (the address of the driver dispatch table) to determine a driver is in memory. Zero and nonzero values for these two pointers have the meanings shown in Figure 4-5.

4.4.3.1 Establishing I/O Function Masks - Table 4-5 is supplied to assist you in determining the proper values to set in the function masks. The mask values are given for each I/O function used by DIGITAL-supplied drivers. The bit number allows you to determine which mask group to use: for bits numbered 0 through 15, use the mask value for a word in the first 4-word group; for bits numbered 16 through 31, use the mask value for a word in the second 4-word group.



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

Figure 4-5: D.PCB and D.DSP Bit Meanings

Of the function mask values listed in Table 4-5, only IO.KIL is mandatory and has a fixed interpretation. However, if IO.ATT and IO.DET are used, they must have the standard meaning. (Refer to the *P/OS System Reference Manual*) for a description of standard I/O functions.) If QIO directive processing encounters a function code of 3 or 4 and the code is not no-op, QIO assumes that these codes represent Attach Device and Detach Device, respectively. The other codes are suggested but not mandatory. You are free to establish all other function-code values on nonfile-structured devices. However, the mask words must still reflect the proper filtering process.

If you are writing a driver for a file-structured device, you must establish the standard function mask values of Table 4-5.

To determine the proper bit masks for disks, tapes, and unit record devices (such as terminals, card readers, line printers, paper tape punches/readers), use Table 4-6, Table 4-7, and Table 4-8 as guides.

Table 4-5: Mask Values for Standard I/O Functions

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

Table 4-6: Mask Word Bit Settings for Disk Drives

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

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

Table 4-7: Mask Word Bit Settings for Magnetic Tape Drives

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

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

Table 4-8: Mask Word Bit Settings for Unit Record Devices

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

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

4.4.4 The Unit Control Block (UCB)

Figure 4-6 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.

The fields* in the UCB are described below:

U.UAB (0)

Driver access:

Initialized, not referenced.

Description:

For terminal UCBs only. Reserved.

U.MUP

Driver access:

Not initialized, not referenced.

Description:

For terminal UCBs only.

U.LUIC (O<100200>)

Driver access:

Initialized, not referenced.

Description:

For terminal UCBs only, and only in multiuser systems: the logon UIC of the user at the particular terminal. This offset must exist for any device on a multiuser system for which the DV.TTY bit is set.

U.OWN (0)

Driver access:

Initialized, not referenced.

---

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

Description:

The UCB address of the owning terminal for allocated devices.

U.UAB[1]	reserved		10	
U.MUP[1]	reserved		6	
U.LUIC[1]	reserved		−4	
U.OWN	Owning terminal UCB address		−2	All devices
U.DCB	Back pointer to DCB		0	
U.RED	Redirect UCB pointer		2	
U.CTL U.STS	Unit status	Control flags	4	
U.UNIT U.ST2	Unit status	Physical unit no.	6	
U.CW1	Characteristics word 1		10	
U.CW2	Characteristics word 2		12	
U.CW3	Characteristics word 3		14	
U.CW4	Characteristics word 4		16	
U.SCB	Pointer to SCB		20	
U.ATT	TCB address of attached task		22	
U.BUF	Buffer relocation bias		24	
U.BUF+2	Buffer address		26	
U.CNT	Byte count		30	
U.UCBX[2]	Pointer to the UCB extension in secondary pool		32	
	Device-dependent		34	
	. . .			
	storage			

1. This offset appears only for terminal devices (that is, devices that have DV.TTY set)
2. This offset appears only for those devices that have DV.MSD set.

Figure 4-6: Unit Control Block

U.DCB (pointer to associated DCB)

Driver access:

Initialized, not referenced.

Description:

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

U.RED (pointer to start of this UCB (.-2))

Driver access:

Initialized, not referenced.

Description:

Contains a pointer to the unit control block to which this device-unit has been redirected. The redirect chain ends when this word points to the beginning of the UCB itself (U.DCB of the UCB, to be precise).

U.CTL (device-dependent values)

Driver access:

Initialized, not referenced.

Description:

U.CTL and the function mask words in the device control block control QIO directive processing. Figure 4-7 shows the layout of the unit control byte.

The driver data base code statically establishes this bit pattern. Any inaccuracy in the bit setting of U.CTL produces erroneous I/O processing. Bit symbols and their meanings are as follows:

UC.ALG - Alignment bit.

If this bit is 0, then byte alignment of data buffers is allowed (for example, a communications driver). If UC.ALG is 1, then buffers must be word-aligned (for example, a disk driver).

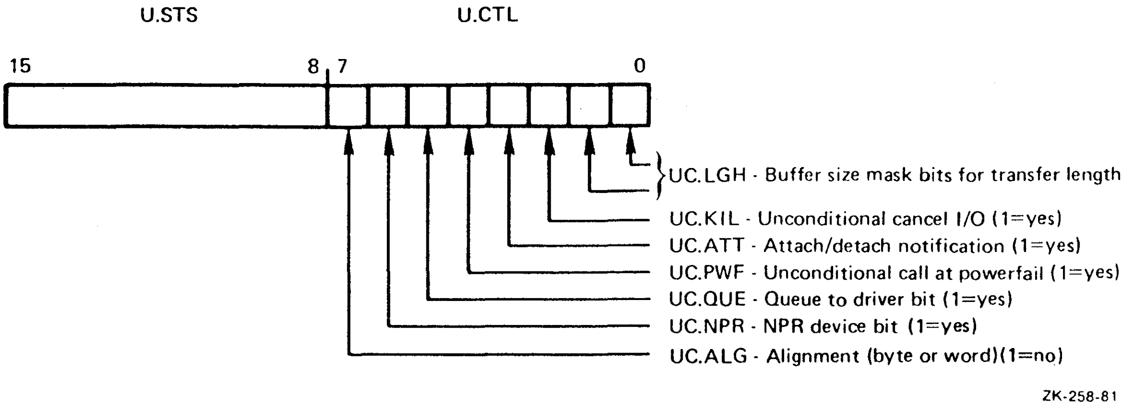


Figure 4-7: Unit Control Byte

UC.ATT - Attach/Detach notification.

If this bit is set, then the driver is called when $GTPKT processes an Attach/Detach I/O function. Typically, the driver does not need to obtain control for Attach/Detach requests, and the Executive performs the entire function without any assistance from the driver. When the need exists, the most common use of UC.ATT is for event notification without outstanding I/O. Attachment coupled with UC.ATT provides an I/O rundown operation (IO.DET) to occur so that the driver can remove this context when the issuing task exits - gracefully or otherwise.

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

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

UC.QUE - Queue to-driver bit.

If set, the QIO directive processor calls the driver at its I/O initiation entry point without queuing the I/O packet. After the processor makes this call, the driver is responsible for the disposition of the I/O packet. Typically, the processor queues an I/O Packet before calling the driver, which later retrieves it by a call to $GTPKT.

The most common reason for a driver to examine a packet before queuing is that the driver employs a special user buffer, other than the normal buffer used in a transfer request. Within the context of the requesting task, the driver must address-check and relocate such a special buffer. See Section 8.1 for an example of a driver that does this. **Use $CKBFR, $CKBFB, OR $CKBFW rather than $ACHRO, $ACHKB, $ACHKW, since the later routines do not increment region and ACB I/O counts.**

UC.PWF - Unconditional call on loading driver bit.

If set and the unit is on-line, the driver is always to be called when driver is loaded. Driver may then ignore unit and controller status changes by simply performing a RETURN instruction. The driver, however, can never be unloaded without reboot if these entry points are ignored for the online to offline transition.

UC.NPR - NPR device bit.

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

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

These two bits are used to check whether the byte count specified in an I/O request is a legal buffer modulus. You select one of the values below by ORing into the byte a 0, 1, 2, or 3.

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

UC.ALG and UC.LGH are independent settings.

U.STS (0)

Driver access:

Initialized, not referenced.

Description:

This byte contains device-independent status information. Refer to the UCBDF$ macro definition in Appendix A. Figure 4-8 shows the layout of the unit status byte.

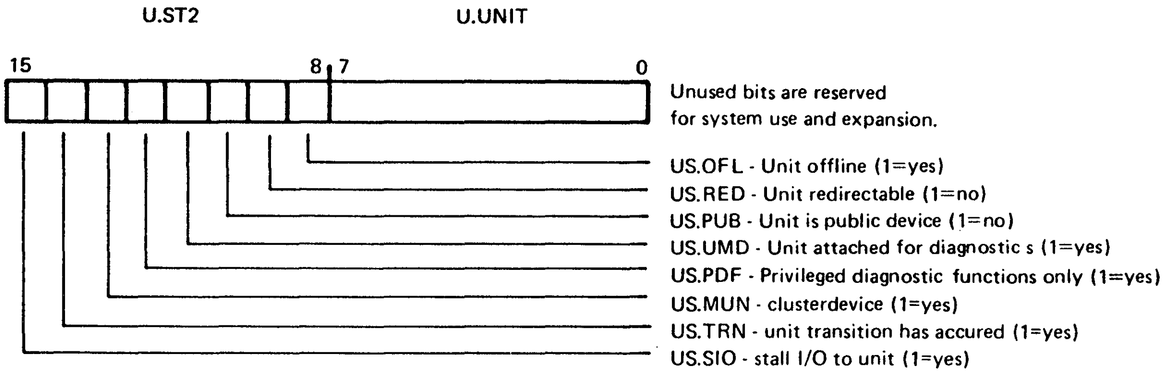


Figure 4-8: Unit Status Byte

US.MDM, US.MNT, US.VV and US.FOR apply only to mountable devices.* 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 mounted foreign.

US.MDM

---

* If your user-written driver services a mountable device, refer to Section 4.5.7 for information on volume valid processing.

If set, device is marked for dismount.

US.VV

If set, volume is valid from a software viewpoint.

U.UNIT (unit number)

Driver access:

Initialized, read-only.

Description:

This byte contains the physical unit number of the device-unit serviced by this UCB. If the controller for the device supports only a single unit, the unit number is always zero.

NOTE

This is the physical unit number of the device and not the logical unit number. The range of this number is from zero to n where n is device-dependent. The logical designation DB0: does not necessarily imply a zero in this byte.

U.ST2 (US.OFL)

Driver access:

Initialized, not referenced.

Description:

This byte contains additional device-independent status information. Different parts of the system set and clear these bits. The layout of the unit status extension byte is shown in Figure 4-9.

The bit meanings are as follows:

US.OFL=1

If set, the device is off-line (that is, not in the configuration). This bit should be initialized to 1.

US.RED=2

If set, the device cannot be redirected.

US.PUB=4

If set, the device is a public device.

US.UMD=10

If set, the device is attached for diagnostics.

US.PDF=20

If set, this unit can be used for a privileged diagnostic function only.

US.TRN=100

If set, unit volume status change in progress.

US.SIO=200

If set, I/O is stalled typically until volume is reverified.

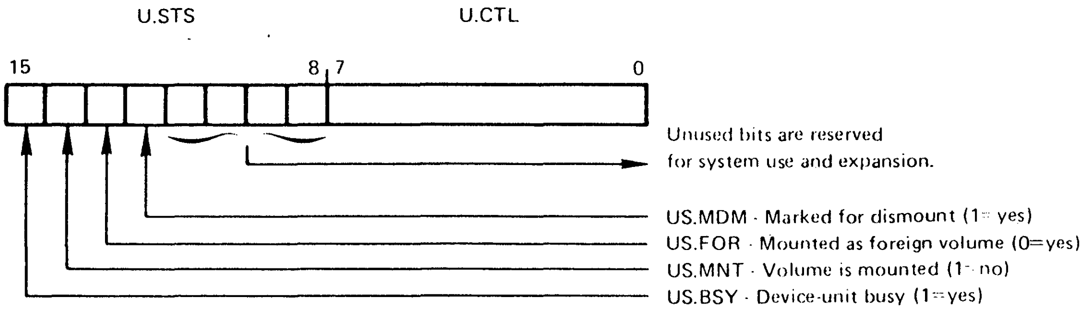


Figure 4-9: Unit Status Extension 2

U.CW1 (device-specific characteristics)

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

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

DV.REC=1

Record-oriented device

DV.CCL=2

Carriage-control device

DV.TTY=4

Terminal device. If DV.TTY is set, then the UCB contains extra cells (for U.LUIC, U.CLI, and optionally U.UAB).

DV.DIR=10

Directory device

DV.SDI=20

Single directory device DV.SQD=40

Sequential device

DV.MSD=100

Mass Storage device

DV.UMD=200

Device supports user-mode diagnostics

DV.EXT=400

Unit is on an extended 22-bit controller

DV.SWL=1000

Unit is software write-locked

DV.ISP=2000

Input spooled device

DV.OSP=4000

Output spooled device

DV.PSE=10000

Pseudo device. If this bit is set, the UCB does not extend past the U.CW1 offset.

DV.COM=20000

Device mountable as a communications channel

DV.F11=40000

Device mountable as a FILES-11 device

DV.MNT=100000

Device mountable*

U.CW2 (device-specific characteristics)

Driver access:

Initialized, read-write.

Description:

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

U.CW3 (device-specific characteristics)

Driver access:

Initialized, read-write.

---

* If your user-written driver services a mountable device, refer to Section 4.5.7 for information on volume valid processing and privileged ACP functions.

Description:

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

U.CW4 (device-specific characteristics)

Driver access:

Initialized, read-only.

Description:

Default buffer size in bytes. This word is changed by a system command (SET with the /BUF keyword). The value in this word effects FCS, RMS, and many utility programs.

U.SCB (SCB pointer)

Driver access:

Initialized, read-only.

Description:

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

U.ATT (0)

Driver access:

Initialized, not referenced.

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

---

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

Description:

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

U.BUF (reserve two words of storage)

Driver access:

Not initialized, read-write.

Description:

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

For transfer operations, the initial format of these two words depends on the setting of UC.NPR in U.CTL. The driver does not format the words; all formatting is completed before the driver receives control. The format is determined by the UC.NPR bit, which is set for an NPR device and reset for a program-transfer device.

The format for program-transfer devices is an address doubleword identically formatted to I.IOSB+2 and I.IOSB+4.

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

The details of the construction of the Address Doubleword appear in Chapter 7.

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 uses this field in constructing the actual device request.

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

U.UCBX

Driver access:

Not initialized, not referenced

Description:

This field contains a pointer to the UCB extension in secondary pool for mass storage devices with DV.MSD set, (DV.MSD=1).

For information on formatting, see the description of the UCBDF$ macro.

U.PRM (Device-dependent words)

Driver access:

Not initialized, read-write.

Description:

The driver establishes this variable-length block of words to suit device-specific requirements. For example, a disk driver uses the first words to store the disk geometry as follows:

```
.BLKB    1     ;# OF SECTORS PER TRACK
.BLKB    1     ;# OF TRACKS PER CYLINDER
.BLKW    1     ;# OF CYLINDERS PER VOLUME
```

The driver can call the $CVLBN routine (described in Chapter 7) to convert a logical block number to a disk address based on the values in U.PRM and U.PRM+2.

4.4.5 The Status Control Block (SCB)

Figure 4-10 is a layout of the SCB. The SCB contains the context for a unit operation and describes the status of a unit that can run in parallel with all other units.

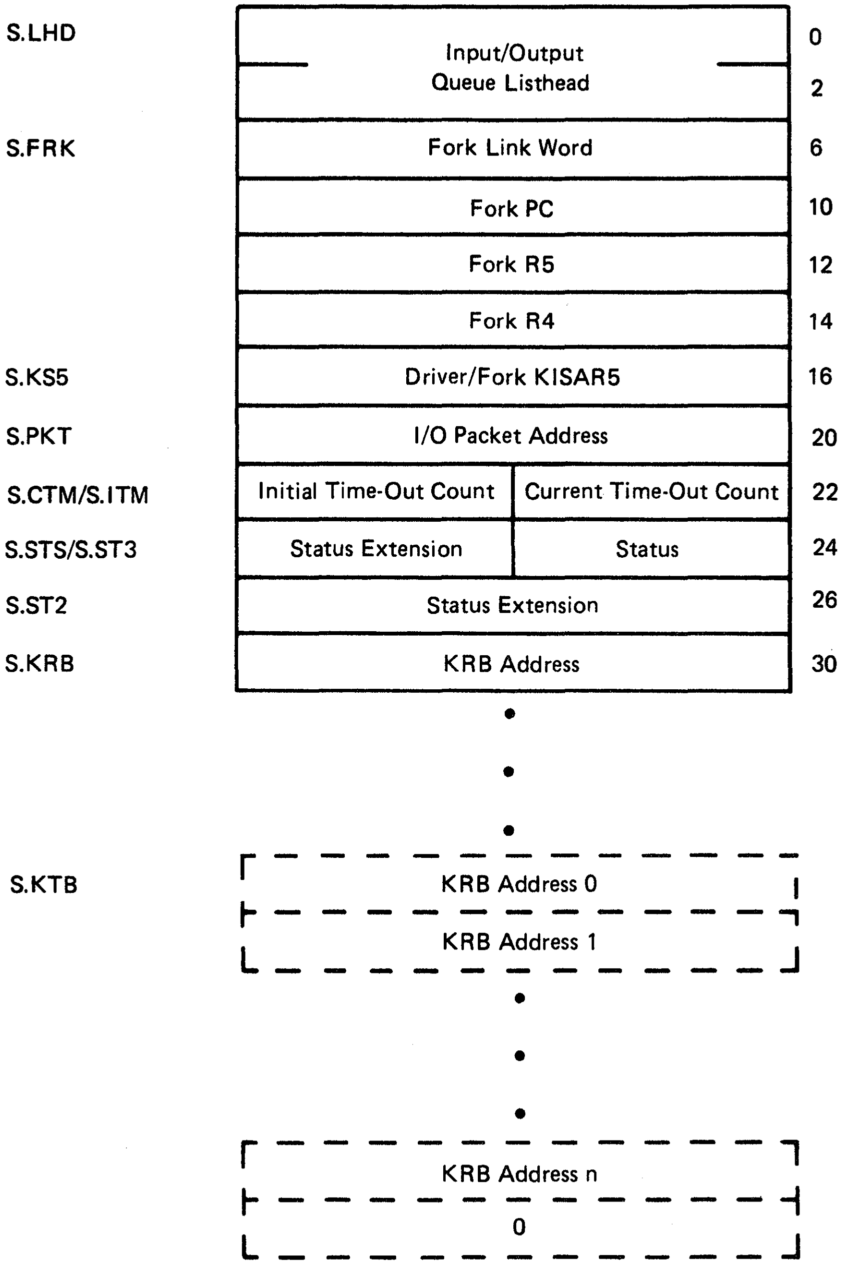


Figure 4-10: Status Control Block

DRIVER DATA STRUCTURE DETAILS

The fields* in the SCB are described as follows:

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

Driver access:

Initialized, not referenced.

Description:

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

S.FRK (reserve four words of storage)

Driver access:

Initialize words to zero, 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.

S.KS5 (0)

Driver access:

Initialized, not referenced.

Description:

This word contains the contents of KISAR5 necessary to correctly alter the Executive mapping to reach the driver for this unit. It is set by PROLOD, and whenever a fork block is dequeued and executed, this word is unconditionally jammed into KISAR5. ADJACENCY WITH THE FORK BLOCK IS ASSUMED!

S.PKT (reserve one word of storage)

---

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

Driver access:

Not initialized, read-only.

Description:

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

S.CTM (0)

Driver access:

Not initialized, read-write.

Description:

P/OS 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 typically initialized by moving S.ITM (initial timeout count) into S.CTM. The Executive clock service (in module TDSCH) examines active times, decrements them, and, if they reach zero, calls the driver at its device timeout entry point.

The internal clock count is kept in 1-second increments. Thus, a time count of 1 is not precise because the internal clocking mechanism is operating asynchronously with driver execution. The minimum meaningful clock interval is 2 if you intend to treat timeout as a consistently detectable error condition. If the count is zero, then no timeout occurs; a zero value is, in fact, an indication that timeout is not operative. The maximum count is 250. The driver is responsible for setting this field. Resetting occurs at actual timeout or within $FORK and $IODON.

S.ITM (initial timeout count)

Driver access:

Initialized, read-only.

Description:

Contains the initial timeout value that the driver can load into S.CTM to begin device timeout.

S.STS (0)

Driver access:

Initialized, not referenced.

Description:

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

S.ST3 (driver-specific status byte)

Driver access:

Initialized, referenced by driver for synchronization.

Description:

This status byte is reserved for driver-specific status bits concerning driver-executive or driver-driver communication. Figure 4-11 shows the layout of this byte.

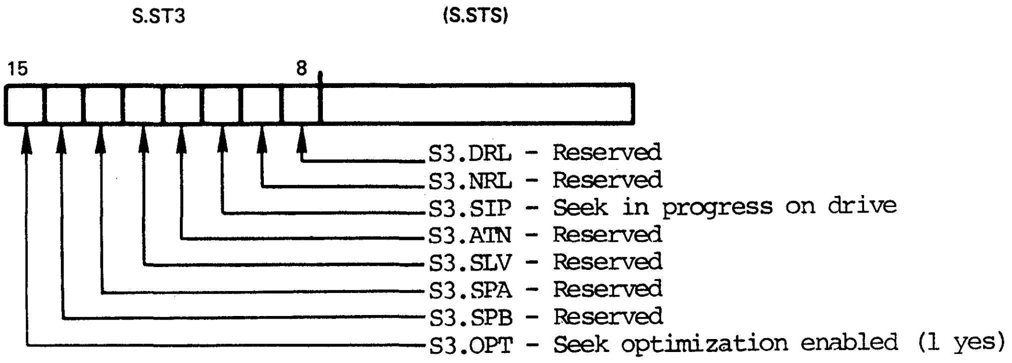


Figure 4-11: Controller Status Extension 3

The following are the descriptions for the currently defined bits. All currently defined bits are used by mass storage devices.

S3.SIP=4

If this bit is set, the drive has a seek in progress. A driver that supports overlapped seek operations examines this bit to keep track of whether the drive is seeking. For a driver that does not support overlapped operations, this bit is set to indicate that a positioning operation is in progress.

S3.SPB=100

If this bit is set, port B on this unit is spinning up.

S3.OPT=200

Reserved for future use. Must be clear.

S.ST2 (controller status extension)

Driver access:

Initialized.

Description:

This status word defines certain status conditions for the controller-unit combination. Figure 4-12 shows the layout of this word.

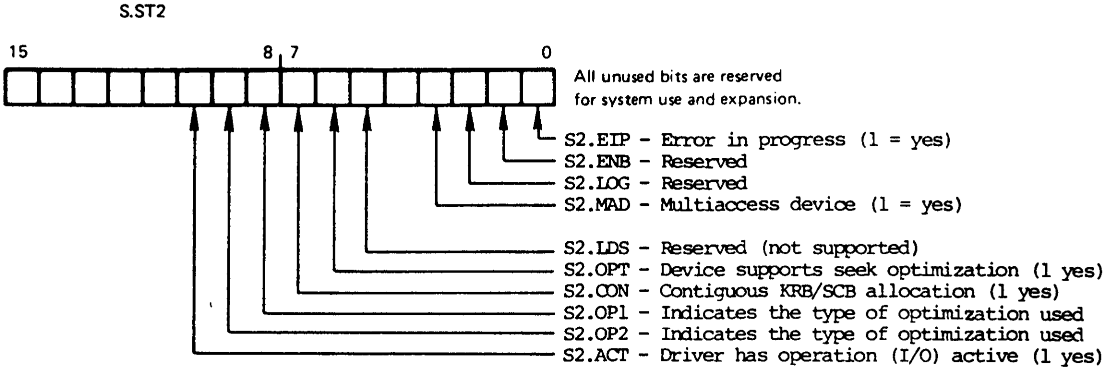


Figure 4-12: Controller Status Extension 2

DIGITAL has attempted to restrict bits in this word to those defining system-wide status. Specific bits for driver and Executive synchronization or driver internal synchronization are allocated from S.ST3. The following are the descriptions for the currently defined bits:

S2.MAD=10

This bit indicates the presence of the table of KRB addresses at the end of the Status Control Block. PROLOD will relocate these KRB addresses, but it is the drivers' responsibility to manage controller assignment.

S2.CON=200

This bit indicates the contiguous allocation of the controller request and status control blocks. Devices that do not support overlapped operation do not require a separate SCB for each unit. The KRB and SCB for such devices can be contiguous and some fields in the SCB overlap those in the KRB. Therefore, the SCB offsets S.CSR, S.PRI, S.VCT, and S.CON are valid only for such devices. For these devices, S2.CON is set.

For the layout of the contiguous KRB and SCB, refer to Section 4.4.7.

S2.ACT=2000

If this bit is set, the driver has active I/O.

S.KRB (pointer to currently assigned KRB)

Driver access:

Initialized, referenced by driver to access the KRB.

Description:

This word points to the currently assigned controller request block. If this word has a value of zero, then the device has no currently assigned KRB. It may, in fact, not have a KRB or CTB at all. Both the null driver and virtual terminal driver have no KRB.

Certain restrictions apply to drivers whose data bases do not include KRBs. They will receive powerfail, timeout, and cancel calls like any other driver, but the priority will always be zero, and the CSR address and controller index (where supplied) will be undefined.

NOTE

All code that checks S.KRB for a KRB pointer must check for a possible zero value and take appropriate action. A zero value in S.KRB does not necessarily mean that a KRB does not exist, but perhaps rather that one is not currently assigned. P/OS systems do not currently provice active support of multi-access devices. If needed, however, the driver may dynamically change this KRB pointer for its own purposes.

The first cell in the KRB (K.CSR) contains the control and device register address for the controller.

S.KTB (KRB addresses)

Driver access:

Initialized.

Description:

This table appears only if and the device is multiaccess (the S2.MAD bit set).

Every controller to which the unit (unit control block and status control block combination) can communicate is represented in this table by a controller request block address. The table contains at least two entries, with the list terminated by a zero word. Only the driver may change S.KRB, and it may or may not use the low-order bit of the KRB addresses in S.KRB as an on-line and off-line flag. System software (other than the driver) must not modify S.KRB and must tolerate a 1 in the low-order bit of the values in S.KTB.

4.4.6 The Controller Request Block (KRB)

Figure 4-13 is a layout of the controller request block. One KRB exists for each controller. If a controller allows only a single operation on a single unit at a time, then the driver can allocate the controller request block and the status control block in contiguous space. With such contiguous allocation, all offsets commonly used by the driver are referenced by their S.xxx forms. The system will still use the offset S.KRB and the K.xxx forms for all references. Refer to Section 4.4.7 for the contiguous SCB/KRB allocation.

The fields* in the KRB are described as follows:

K.PRM (device-dependent storage)

Driver access:

Initialized, read-write.

Description:

PROLOD does not relocate any addresses in this area.

---

* Parenthesized comments following the symbolic offset indicate the value to be initialized in the data base source code.

K.PRM	Driver dependent storage		
K.SLT	interrupt controller CSR		
K.ICSR	reserved	slot number	
K.VCT^5/ K.PRI^5	Vector/4	Priority	– 6
K.IOC/K.CON^5	Controller I/O count	Controller index	– 4
K.STS	Controller status		– 2
K.CSR^5	Control and status register address		0
K.OFF	Offset to UCB table		2
K.HPU	Unused	Highest physical unit	4
K.OWN	Owner (UCB address of unit owned)		6
K.CRQ^1	Controller request queue listhead		10
K.URM^{1,2}	Controller UNIBUS run mask		14

•
•
•

Start of UCB table

4	UCB address physical unit 0
	• • •
4	UCB address physical unit n
4	-1

Figure 4-13: Controller Request Block

K.PRI (device priority)

Driver access:

Initialized, read-only.

Description:

Contains the priority at which the device interrupts. Use symbolic values (for example, PR4) to initialize this field in the driver data source code. These symbolic values are defined by issuing the HWDDF$ macro.

K.VCT (interrupt vector divided by 4)

Driver access:

Initialized, not referenced.

Description:

First interrupt vector address divided by 4.

K.CON (controller number times 2)

Driver access:

Initialized, read-only.

Description:

Controller number multiplied by 2. Drivers that support more than one controller use this field. A driver may use K.CON to index into a controller table created in the driver data base source code and maintained internally by the driver itself. By indexing the controller table, the driver can service the correct controller when a device interrupts.

Because this number is an index into the table of addresses in the CTB, its maximum value is limited by the value of L.NUM in that CTB.

K.IOC (0)

Driver access:

Initialized, not referenced.

Description:

Reserved for future use.

K.STS (controller-specific status)

Driver access:

Initialized, not referenced.

Description:

This word is used as a status word that concerns the controller. Figure 4-14 shows the layout of the controller status word.

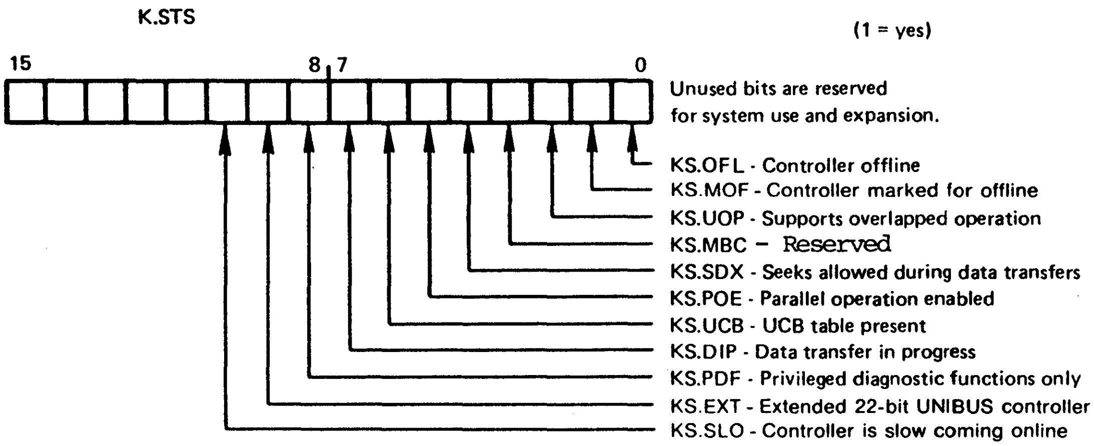


Figure 4-14: Controller Status Word

All undefined bits are reserved for use by DIGITAL. Currently defined bits are:

KS.MOF=2

If this bit is set, the unit/controller is in the process of becoming offline.

KS.UOP=4

This bit indicates whether the controller supports unit operation in parallel and requires synchronization. If this bit is set, each unit attached to the controller is capable of operating independently. Therefore, the KRB contains a UCB table holding the UCB addresses of each independent unit.

KS.SDX=20

If this bit is set, the controller allows seek operations to be initiated while a data transfer is in progress. (Some types of disks, such as the RK06 and RK07, support overlapped seek operations but do not allow a seek to be initiated if a data transfer is in progress.) The Executive routines Request Controller for Control Function ($RQCNC) and Request Controller for Data Transfer ($RQCND) examine this bit to distinguish between the two types of controllers that support overlapped seeks.

KS.POE=40

If this bit is set, the driver may initiate an I/O operation on the controller in parallel with other I/O operations. A driver that supports overlapped seek operations checks this bit to decide whether it should attempt to perform an I/O operation as a seek phase and then a data transfer phase (that is, overlapped) or as an implied seek (that is, nonoverlapped). If this bit is set, the driver can then attempt the overlapped operation.

An overlapped driver must check this bit once only for each I/O operation. The driver must not rely on the bit value to decide whether, upon being interrupted, the driver was attempting a seek operation. The driver should use the S2.SIP bit to hold its internal state.

KS.UCB=100

This bit indicates the presence of the table of unit control block addresses associated with the KRB. If this bit is set, K.OFF gives the offset from the beginning of the KRB to the start of the UCB table.

Devices that support unit operation in parallel (for example, overlapped seeks) require a mechanism for finding the UCB of the unit generating an interrupt. Therefore, if KS.UOP is set, a UCB table must exist. If KS.UOP is not set, however, a UCB table may still exist because some devices (for example, terminal multiplexers) support full unit operation in parallel but do not require synchronization. Therefore, KS.UCB may be used to determine whether the UCB table exists, regardless of whether KS.UOP is set.

KS.DIP=200

If this bit is set, a data transfer is in progress. A driver that supports overlapped seek operation sets or clears this bit to indicate to itself whether, after an interrupt, a data transfer is in progress. The driver must set or clear this bit. Usage of this bit eliminates the need for the software to access the device registers to determine what type of operation was in progress.

K.CSR (start of controller device register addresses)

Driver access:

Initialized, read-only.

Description:

Contains the first address of the device for the device controller. The driver uses K.CSR to initiate I/O operations.

NOTE

This word is guaranteed to be offset zero for the KRB.

K.OFF (offset in bytes (from K.CSR) to start of UCB table)

Driver access:

Initialized, referenced by interrupt dispatch code.

Description:

This word contains the offset to the beginning of the unit control block table. When added to the starting address of the KRB, it yields the UCB table address.

The status bit KS.UCB may be used to determine whether the UCB table exists. A UCB table may exist if KS.UOP is not set, since some devices (for example, terminal multiplexers) support full unit operation in parallel with no synchronization required. If KS.UOP is set, a UCB table must appear (and KS.UCB will also be set).

K.HPU (highest physical unit number)

Driver access:

Initialized.

Description:

This byte contains the value of the highest physical unit number used on this controller.

K.OWN (0)

Driver access:

Initialized, referenced for actual unit.

Description:

This word has three slightly different uses, depending on the particular device.

1. For controllers which always have only a single unit connected to them (for example, the line printer), K.OWN/S.OWN always points to the UCB of that unit. You can use the suc argument in the GTPKT$ macro to statically initialize this cell in the data base.

2. For controllers that may have multiple units attached but do not support unit operation in parallel, K.OWN/S.OWN is set with the currently active unit by code generated with the GTPKT$ macro suc argument set to blank.

3. For controllers that support unit operation in parallel and require synchronization (KS.UOP is set), this is a busy/nonbusy interlock for the controller. If the controller is busy for a data transfer, this word contains the UCB address of the currently active unit. This word is set and cleared by the Request Controller for Control Access ($RQCNC), Request Controller for Data Access ($RQCND), and Release Controller ($RLCN) routines.

K.CRQ (first word equals 0; second word points to first)

Driver access:

Initialized, not referenced.

Description:

Two words that form the controller wait queue. Fork blocks are queued here for driver processes that have requested controller access. Driver processes that request access for control functions are queued on the front of the list, and those that request access for data transfer are queued on the end of the list.

KE.UCB

Driver access:

Initialized, referenced by interrupt dispatch code.

Description:

This table contains the unit control block addresses for the units on this controller. Physical unit zero is in the first word, unit one is in the second word, and unit n is in word n+1. The table has a length of (K.HPU+1) words. A value of zero in this table indicates a physical unit number for which no actual physical unit exists. The table is terminated by a -1.

NOTE

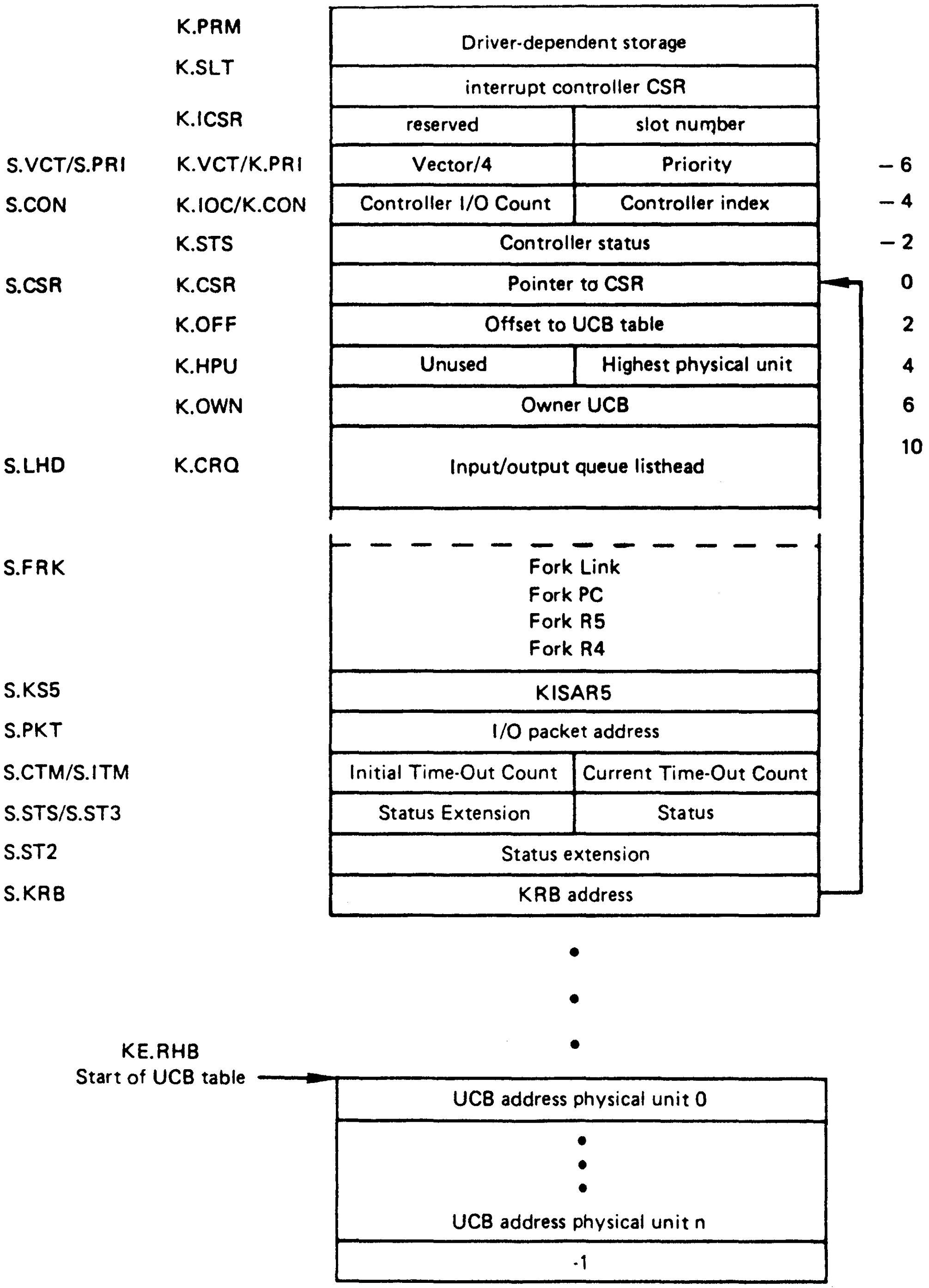
This table exists only for those devices that have KS.UCB set.

4.4.7 Contiguous Allocation of the SCB and KRB

In a configuration where a controller and the Executive supports only a single operation on a unit at one time, the driver can allocate space for the KRB and the SCB in a contiguous area. Some fields of the KRB overlap those in the SCB. Although the KRB and SCB in this arrangement are contiguous, the system still considers the I/O data structure to contain a KRB. The system will still use the S.KRB offset and the K.xxx forms for all references. The driver can reference the fields by the S.xxx form of the symbolic offset definitions. Figure 4-15 shows the physical layout of the contiguous KRB and SCB allocation.

4.4.8 Controller Table (CTB)

Figure 4-16 is a layout of the controller table. You ensure that the CTB is linked into the system list of controller tables by placing the global label $xxCTB at the start of the table of the KRB addresses in the CTB.

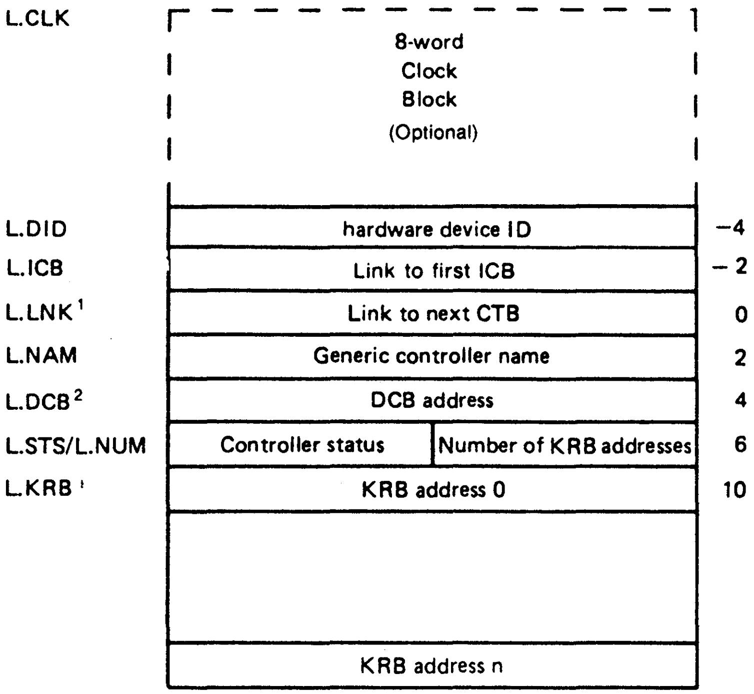


	K.PRM	Driver-dependent storage		
	K.SLT	interrupt controller CSR		
	K.ICSR	reserved	slot number	
S.VCT/S.PRI	K.VCT/K.PRI	Vector/4	Priority	–6
S.CON	K.IOC/K.CON	Controller I/O Count	Controller index	–4
	K.STS	Controller status		–2
S.CSR	K.CSR	Pointer to CSR		0
	K.OFF	Offset to UCB table		2
	K.HPU	Unused	Highest physical unit	4
	K.OWN	Owner UCB		6
S.LHD	K.CRQ	Input/output queue listhead		10
S.FRK		Fork Link Fork PC Fork R5 Fork R4		
S.KS5		KISAR5		
S.PKT		I/O packet address		
S.CTM/S.ITM		Initial Time-Out Count	Current Time-Out Count	
S.STS/S.ST3		Status Extension	Status	
S.ST2		Status extension		
S.KRB		KRB address		

KE.RHB
Start of UCB table

UCB address physical unit 0
• • • UCB address physical unit n
-1

Figure 4-15: Contiguous KRB/SCB Allocation



---

[1] The head of the list of controller tables is $CTLST in SYSCM.

[2] If LS.CIN is set, this cell points to the common interrupt address table rather than to the DCB.

Figure 4-16: Controller Table

The fields* in the CTB are described below:

L.CLK

Driver access:

Initialized

Description:

---

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

This is the clock queue entry for these devices that need a single clock block per generic controller type. It only appears if LS.CLK is set.

L.ICB (reserve one word of storage)

Driver access:

Not initialized, not referenced.

Description:

This word contains the address of the first interrupt control block for this type of controller, divided by two (2).

L.LNK (0 or link to next CTB in list)

Driver access:

Not initialized, not referenced.

Description:

All the controller tables in the system are linked together so they can be found, and they are threaded through this first word. A zero link terminates this list.

A CTB must exist for every physical controller type in the system.

L.DID (controller's hardware ID)

Driver access:

Initialized, read-only.

Description:

This hardware ID is the controller mnemonic used to find this controller table from among all the others in the system.

L.NUM (number of KRB addresses)

Driver access:

Initialized, read only.

Description:

Used by programs that scan the controller tables to compute the number of KRB addresses. This value is never zero, since without controller request blocks there should be no controller table.

L.STS (generic controller status)

Driver access:

Initialized, read only.

Description:

The controller table status bits give information about the class of controllers. Figure 4-17 shows the layout of this byte.

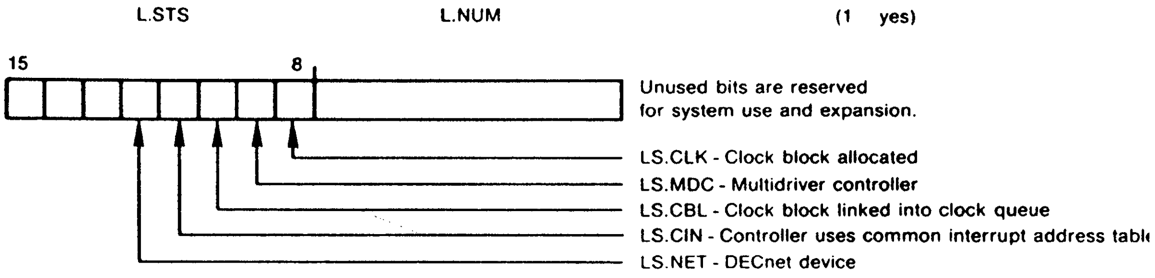


Figure 4-17: Controller Table Status Byte

The following are the descriptions of these bits:

LS.CLK=1

If this bit is set, the controller table has an 8-word clock block.

LS.MDC=2

If this bit is set, multiple drivers service units attached to the associated controller.

LS.CBL=4

If this bit is set, the clock block is linked into the clock queue.

L.KRB (KRB addresses of controllers)

Driver access:

Initialized once for the controller, not referenced.

Description:

A list of the controller request block addresses ordered by their respective controller numbers. This table is indexed by the controller index retrieved from the PS word immediately after an interrupt. The table is of length (L.NUM) words. While the interrupt routines will not have to scan the list in a linear fashion, the only way to find all the controller request blocks in the system includes a linear scan of all the controller tables. The CTB is static.

The address of the start of the KRB address list in the CTB is the global symbol $xxCTB in the driver dispatch table. PROLOD supplies this address in the DDT when it loads the driver.

Proper action for drivers to access their list of KRB addresses is to retrieve the address of the start of the KRB list in the CTB from the cell in the driver dispatch table set up by PROLOD.

4.5 DRIVER CODE DETAILS

This section describes the specific requirements for driver code. The driver code must contain a driver dispatch table which allows the Executive to call the driver to perform discrete system functions. If the driver needs to access either system structures such as the partition and task control blocks or structures within its own data base, it should use the system-wide symbolic offsets rather than the real offsets. Because the driver is built with the Executive library EXELIB.OLB, the symbolic offsets are automatically defined for the driver code. If you want to see the definitions of the symbols in your driver listing, place in your driver source code the related macro name in a .MCALL directive and invoke the macro. (For your convenience, the source code of the macro calls that define the symbols of structures is in Appendix A.) The detailed descriptions of the driver data base structures are in Section 4.4.

4.5.1 Driver Dispatch Table Format

The driver dispatch table associates the entry points that the Executive expects to find in a device driver and the actual locations of the routines in the driver code. The DDT also provides a link from the driver code to the driver data base. Figure 4-18 shows the format of the DDT. Section 4.3.1 describes the DDT$ macro call, which automatically generates the DDT.

All device drivers require a driver dispatch table somewhere in the first 4K words of the driver code. Conventionally, the table is located at the beginning of the code.

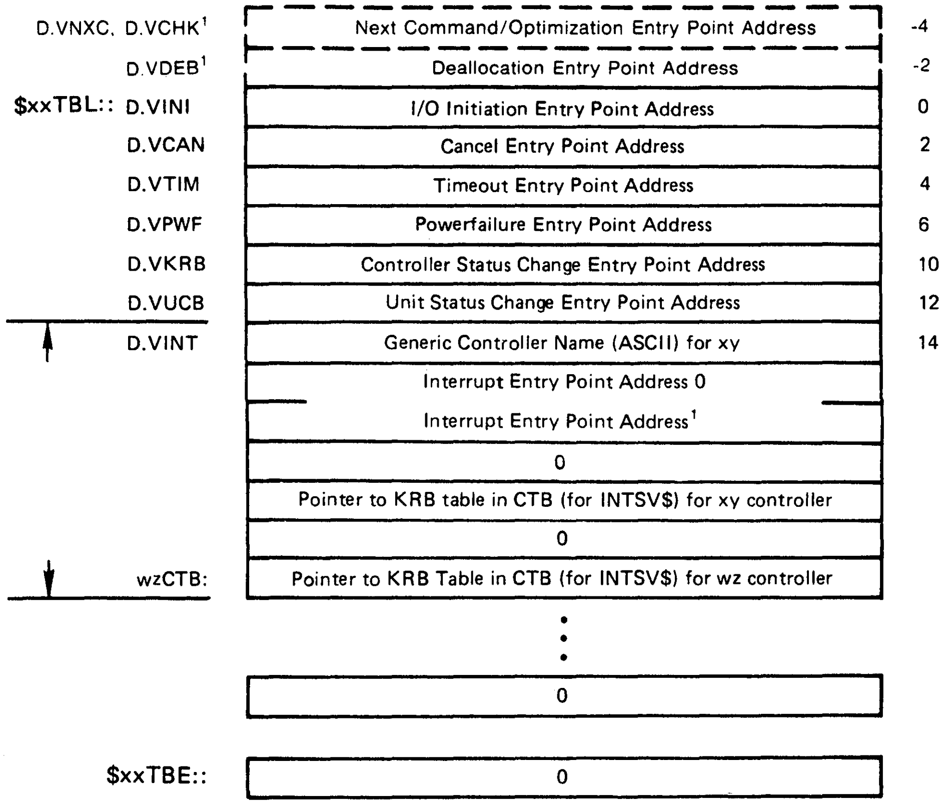
NOTE

> If the length of a driver must exceed 4K words (20000 octal bytes), then your driver must set up the mapping for the second 4K words whenever it is entered; and, of course, all entry points must be in the first 4K words of the driver.

The driver must define some labels that the Executive routines and the INTSV$ macro call use to access the DDT. Table 4-10 lists these labels, which are automatically generated by the DDT$ macro call. Because these labels do not appear in the DDT itself, their format is fixed and they must be specified in the format shown.

Table 4-9: Labels Required for the Driver Dispatch Table

Required Format	Meaning
$xxTBL::	Defines the start of the DDT. The PROLOD routines use this label to fill in D.DSP.
xxCTB:	Defines the pointer to the table of KRB addresses in the CTB of the controller for device xx. Because a driver can support different types of controllers, there may be more than one of this form of label. (The DDT$ macro supports only one controller type.)
$xxTBE::	Defines the end of the DDT for Executive PROLOD routines that scan the DDT.



D.VNXC, D.VCHK[1]	Next Command/Optimization Entry Point Address	-4
D.VDEB[1]	Deallocation Entry Point Address	-2
$xxTBL:: D.VINI	I/O Initiation Entry Point Address	0
D.VCAN	Cancel Entry Point Address	2
D.VTIM	Timeout Entry Point Address	4
D.VPWF	Powerfailure Entry Point Address	6
D.VKRB	Controller Status Change Entry Point Address	10
D.VUCB	Unit Status Change Entry Point Address	12
D.VINT	Generic Controller Name (ASCII) for xy	14
	Interrupt Entry Point Address 0	
	Interrupt Entry Point Address[1]	
	0	
	Pointer to KRB table in CTB (for INTSV$) for xy controller	
	0	
wzCTB:	Pointer to KRB Table in CTB (for INTSV$) for wz controller	
	⋮	
	0	
$xxTBE::	0	

[1] These are optional advance driver features

Figure 4-18: Driver Dispatch Table Format

At offsets D.VINI through D.VUCB in the DDT of your driver appear labels defining the addresses of the entry points in the driver. As a standard procedure, you supply the labels described in Table 4-10 at the entry points in the driver code. The formats of the standard labels that appear in the DDT are not fixed. Because the Executive expects to find the entry point addresses at fixed offsets from the start of the DDT and the labels themselves appear in the DDT, you can change their format if you construct the DDT without using the DDT$ macro call. However, other labels that are required in the driver code but do not appear in the DDT have a certain, fixed format which you must not change. For reference, these fixed format labels are the following:

```
$xxTBL::
$xxTBE::
$xxLOA::
$xxUNL::
```

These fixed-format labels are described elsewhere in this chapter. The DDT$ macro uses the standard labels but allows you to alter the format of some of them.

At offset D.VINT in the DDT is the name of the controller type that the driver supports. (The same name is in the CTB.) If the driver has no controller (such as the virtual terminal driver VTDRV), this word is zero. The structure allows the driver to support multiple controller types. (The terminal driver supports different controller types.) Although the DDT$ macro supports only one controller type, there is no restriction on the number of controller types that a driver can support.

After each controller name follows a block of interrupt entry addresses. At location D.VINT+2 begins the first interrupt address block, each word of which defines an address to be included in a vector for the driver. A zero terminates the block and indicates that there are no more interrupt entry points for the controller. There is no restriction on the number of vectors each controller may have. For a single interrupt device, location D.VINT+2 (interrupt entry address 0) is the interrupt address.

Table 4-10: Standard Labels for Driver Entry Points

Label	Entry Point
xxINI:	I/O initiation
xxCAN:	Cancel I/O
xxCHK:	Block check and conversion
xxOUT:	Device timeout
xxPWF:	Power failure
xxKRB:	Controller status change
xxUCB:	Unit status change
$xxINT:	Interrupt entry point

* The characters xx are the 2-character mnemonic.

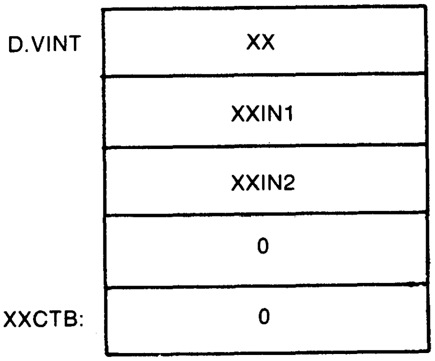


Figure 4-19: Sample Interrupt Address Block in the DDT

4.5.2 I/O Initiation Entry Point

The offset D.VINI in the driver dispatch table contains the address of this entry point. A driver is called at this entry point at priority 0 from the Executive routine $DRQRQ in the module DRSUB. A driver should call the Executive $GTPKT routine to get an I/O packet to process. This action dequeues an I/O request. The following are the register conventions when the Executive enters the driver.

R5 = address of the UCB of the unit for which the Executive has queued an I/O packet

This entry condition pertains unless the driver wants to delay the queuing operation. Therefore, if the queue-to-driver bit UC.QUE in the unit status block offset U.CTL is set, the following are the register conventions.

R5 = UCB address of unit for which a packet has been created
R4 = SCB address of the related unit
R1 = address of the I/O packet

For information on coding requirements for the queue-to-driver operation, see the description of the UC.QUE bit in Section 4.4.4. See Chapter 8 for an example of its use.

The GTPKT$ macro call automatically generates the call to the $GTPKT routine and the code to process the return from $GTPKT. Upon return from $GTPKT, the C bit indicates whether there is a packet to process.

C = 1	If the C bit is set, the Executive found the controller busy, could not dequeue a request, or had to call $FORK to have the driver run on the correct processor.
C = 0	If the C bit is clear, the Executive successfully dequeued a packet for the driver and placed it in the device's input/output queue.

If a request was successfully dequeued, the following are the contents of the registers:

R5 = Address of unit control block
R4 = Address of status control block
R3 = Controller index
R2 = Physical unit number of device to process
R1 = Address of the I/O packet

If the C bit is set, the driver returns control to the caller (a RETURN instruction should be executed). If the C bit is clear, the generated code loads the location at offset K.OWN/S.OWN in the contiguous KRB/SCB with the UCB address of the unit to process. The driver may then process the request and activate the device. All registers are available to the driver. The driver executes a RETURN instruction to transfer control to the system.

4.5.3 Cancel Entry Point

The offset D.VCAN in the driver dispatch table contains the address of this entry point. The Executive routine $IOKIL in the IOSUB module calls the driver at this entry point at device priority. When the Executive enters the driver, the following register conventions pertain:

R5 = UCB address
R4 = SCB address
R3 = Controller index (undefined if S.KRB equals zero)
R1 = Address of TCB of current task
R0 = Address of active I/O packet

The usage of this entry point is explained in Section 2.2.2. All registers are available to the driver. The driver returns control to the Executive by executing a RETURN instruction.

4.5.4 Device Timeout Entry Point

The offset D.VTIM in the driver dispatch table contains the address of this entry point. Routines in the Executive module TDSCH call the driver at this entry point at device priority. When the Executive enters the driver, the entry conditions are as follows:

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

The usage of this entry point is explained in Section 2.2.3. All registers are available to the driver. The driver returns control to the Executive by executing a RETURN instruction.

4.5.5 Deallocation Entry Point

The offset D.VDEB in the driver dispatch table contains the address of this entry point. This entry point is called at priority zero from the routine $FINBF in the Executive module SYSXT after a buffered I/O request completes. The driver is expected to deallocate its buffers at this entry point. When called, the registers are set up as follows:

R0 = address of the first buffer

All registers are available to the driver. The driver returns control to the Executive by executing a RETURN instruction.

4.5.6 Power Failure Entry Point

The offset D.VPWF in the driver dispatch table contains the address of this entry point. The routines in the Executive module POWER call the driver at this entry point at priority 0 for both unit and controller power failures. The Executive first calls the driver for controller power failure with the C bit set. The driver is called in this fashion once for each controller. The following are the register conventions:

C bit set (controller power failure)

R3 = CTB address
R2 = KRB address

The driver may use all registers.

After the Executive has called the driver for all related controllers, it calls the driver once for each unit power failure at priority 0 with the C bit clear. The following are the register conventions:

C bit clear (unit power failure)

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

For both controller and unit power failures, the driver returns control to the calling routine by executing a RETURN instruction.

4.5.7 Controller Status Change Entry Point

The offset D.VKRB in the driver dispatch table contains the address of this entry point. The Executive routine $KRBSC in the OLRSR module calls the driver at this entry point at priority 0 to put a controller on-line or to take a controller off-line.
The C bit indicates whether the request is for off-line or on-line. The following are the register conventions upon entry to the driver.

R3 = CTB address for the controller

R2 = KRB address of controller changing status
0(SP) = Return address for completion
2(SP) = Return address for caller of the Executive routine

The C bit is set to indicate the requested status change as follows:

C = 1 On-line to off-line transition
C = 0 Off-line to on-line transition

The status change byte $SCERR is preset as follows:

$SCERR = 1

The driver indicates the return status in the $SCERR byte as follows:

$SCERR < 0	Operation is not successful and a negative value in $SCERR is the I/O error code. Thus, a negative value rejects the status change requested by the C bit.
$SCERR = 1	Operation is successful. The driver accepts the status change requested. This is the default condition.

All registers are available to the driver. The Executive does not change the status of the controller until and unless the driver shows successful completion of the on-line or off-line request.

The driver must return immediately by either of the following methods:

1. The driver can indicate the return status immediately and can return to the first address on the stack in the normal fashion. If the driver accepts the status change, it merely executes a RETURN instruction. (The status change byte $SCERR has been preset with 1.) If the driver rejects the status change, it loads the relevant I/O error code into $SCERR and executes a RETURN instruction.

2. The driver need not indicate the status immediately but removes the first address from the stack, saves it, and returns immediately to the second address. The driver then has 60 seconds to perform its processing, to indicate the return status, and to return to the first address. The driver can use the offset S.CTM in the status control block to time out some operation (such as a protocol rundown) and then accept or reject the operation by using $SCERR.

If the driver does not return to the first address on the stack, the system can be considered to be in an indeterminate state and possibly corrupted. The driver must return immediately because status changes should not stall the system. The 60-second delay allows a driver time to overcome conditions over which it has little control (such as network connections). System disk and terminal drivers must indicate return status immediately.

4.5.8 Unit Status Change Entry Point

The offset D.VUCB in the driver dispatch table contains the address of this entry point. The Executive routine $UCBSC in the OLRSR module calls the driver at this entry point at priority 0 to put a unit on-line or to take a unit off-line. This entry is called once for each unit whose status changes. The C bit indicates whether the request is for on-line or off-line. The following are the register conventions:

```
     R5 = Address of UCB or unit changing status
     R4 = Address of SCB of unit
     R3 = Controller index (undefined if S.KRB equals zero)
  0(SP) = Return address for driver completion
  2(SP) = Return address for caller of the Executive routine
```

The C bit is set to indicate the requested status change as follows:

C = 1 On-line to off-line transition
C = 0 Off-line to on-line transition

The status change byte $SCERR is preset as follows:

$SCERR = 1

The driver indicates the return status in the $SCERR byte as follows:

$SCERR < 0	Operation is not successful and a negative value in $SCERR is the I/O error code. Thus, a negative value rejects the change requested by the C bit.
$SCERR = 1	Operation is successful. The driver accepts the status change requested. This is the default condition.

All registers are available to the driver. The driver must return within 60 seconds. The Executive does not change the status of a unit until and unless the driver shows successful completion of the on-line or off-line request.

The driver must return immediately by either of the following methods:

1. The driver can indicate the return status immediately and can return to the first address on the stack in the normal fashion. If the driver accepts the status change, it merely executes a RETURN instruction. (The status change byte

$SCERR has been preset with 1.) If the driver rejects the status change, it loads the relevant I/O error code into $SCERR and executes a RETURN instruction.

2. The driver need not indicate the status immediately but removes the first address from the stack, saves it, and returns immediately to the second address. The driver then has 60 seconds to perform its processing, to indicate the return status, and to return to the first address. The driver can use the offset S.CTM in the status control block to time out some operation (such as a protocol rundown) and then accept or reject the operation by using $SCERR.

If the driver does not return to the first address on the stack, the system can be considered to be in an indeterminate state and possibly corrupted. The driver must return immediately because status changes should not stall the system. The 60-second delay allows a driver time to overcome conditions over which it has little control (such as network connections). System disk and terminal drivers must indicate return status immediately.

4.5.9 Interrupt Entry Point

Upon an interrupt, control is dispatched to the driver from an interrupt vector through an interrupt control block or directly from an interrupt vector. A device may have more than one interrupt entry point. The entries in the DDT interrupt address block are used to initialize either the vector(s) or the interrupt control block with the address(es) of the related interrupt entry point(s). (Refer to Section 4.5.1 for a discussion of the interrupt address block.) All drivers should observe the protocol for handling interrupts introduced in Section 1.3 and summarized in Section 4.1.

The driver will be called from the interrupt dispatch coroutine $INTSI in the Executive. The following are the register contents when the driver gets control:

R4 = Controller index

Registers R4 and R5 are available to the driver. The driver runs at the priority set in the interrupt control block. To dismiss the interrupt, a driver executes a RETURN instruction.

Drivers should use the INTSV$ macro call at an interrupt entry point, in order to resolve entry processing. INTSV$ does not generate a call to $INTSV because PROLOD establishes in the interrupt control block the call to the $INTSI coroutine. The $INTSI coroutine saves R4 and R5; sets the priority to that in the interrupt control block; and forms the controller index from the PS and stores it in R4.

INTSV$ generates code to load R5 with the UCB address of the interrupting unit. After the INTSV$ call in the driver code, the following conditions apply:

R5 = UCB address of the interrupting unit
R4 = Controller index

The driver may then do the following:

1. Save extra registers if necessary
2. Do whatever processing is necessary
3. Become a fork process to access the data structures or to call Executive routines if necessary
4. Restore the explicitly saved extra registers
5. Execute a RETURN instruction to the coroutine, which dismisses the interrupt

4.5.10 Volume Valid Processing

System-supplied drivers that service mountable devices (those that have the DV.MNT bit in the UCB U.CW1 word set) take advantage of special processing of volume valid for a device. For such devices the Executive directive processor DRQIO checks that either of the mounted status bits US.MNT or US.FOR in the UCB U.STS word is set. If a mounted status bit is not set, DRQIO requires that a device-specific bit called volume valid (US.VV) be set or else it rejects the directive. If a mounted status bit is set, DRQIO does not check the volume valid bit. (DRQIO assumes that the MOUNT command properly set the volume valid bit.)

To effectively service a mountable device on the system, a user-written driver should perform in one of two ways. First, it can take advantage of the volume valid capability in the same way that a system-supplied driver does. This processing involves calling the $VOLVD routine in the Executive module IOSUB, and handling the spinning-up status bit (US.SPU) and the volume valid bit (US.VV) in the UCB status byte U.STS. (For details of this mechanism, refer to driver source code supplied on the system.) Second, a user-written driver can circumvent the volume valid processing by doing the following:

DRIVER CODE DETAILS

1. Enable the set characteristics function (IO.STC) for volume valid in the DCB legal function mask word
2. Enable the same function in the DCB no-op function mask work
3. Statically set the US.VV bit in the UCB in the driver data base source code

The second method allows the device to be successfully mounted and associated with an ancillary control processor without your having to include code in the driver to handle US.VV.

CHAPTER 5

INCORPORATING A USER-SUPPLIED DRIVER INTO P/OS

This chapter describes how to incorporate a user-supplied driver into an P/OS system. The material in the chapter depends on your having created source code according to the programming specifics given in Chapter 4.

5.1 INCORPORATING AN I/O DRIVER INTO A P/OS SYSTEM

With the exception of DIGITAL supplied disk I/O drivers and the terminal driver, all P/OS I/O drivers are loadable with a loadable data base and are incorporated directly into a running system via a call to the PROLOD (POSSUM) system service. The driver may also be unloaded at runtime, as a result of a specific unload request to PROLOD. The driver data base is never removed as there are many structures in the system which reference the UCB. Even if it were possible to track down all references, the required structures might not be memory resident. Since the system image is not modified as a result of loading a new driver, the data base can be removed from the system by rebooting P/OS.

5.1.1 Guidelines for Creating/Adding a Driver Into the System

To incorporate a loadable driver with a loadable data base, use the following procedure:

1. Create the driver's macro source code file and the data base source code file in the directory of your choice.

2. Assemble the driver and data base using the prefix file RSXMC.MAC and the executive data structures macro library EXEMC.MLB. RSXMC contains the various system configuration symbols used by the executive's conditionalization and commonly used macros, such as DDT$, GTPKT$, and INTSV$.

3. Taskbuild (using PAB) your driver code and data base. Ensure that a symbol table file (.STB) is generated at this time, since it is required by the PROLOD system service. The symbol table file is expected to be located in the same directory and have the same file name as the driver image file (.TSK). The only difference will be the file name extension.

4. Create a program that will issue the PROLOD system service call to load your driver. As an aid to debugging, the logical "PROLOD$MSG" (without the quotes) may be created using the DCL "ASSIGN" command and set to ASCII "0". This logical's existence will enable PROLOD ASCII error messages to be sent to the debugging terminal (TT2:).

5.1.2 Assembling the I/O Driver

After you have created the driver source code and data base files, they must be assembled by the P/OS macro assembler (PMA). It is suggested that listing files be created in the event debugging is needed. The following commands illustrate this:

```
PMA>DRVCOD,DRVCOD=[1,5]EXEMC/ML,RSXMC/PA:1,[]DRVCOD
    ; Assemble the driver code.
PMA>DRVTAB,DRVTAB=[1,5]EXEMC/ML,RSXMC/PA:1,[]DRVTAB
    ; Assemble the driver data base.
```

Note that it is not possible to use ODT in a driver since a driver is not a task, and it is not possible to issue a system directive from system state. For debugging tips and procedures, see Chapter 6.

5.1.3 Taskbuilding the I/O Driver

After completing the assemblies with no detected errors, taskbuild the driver code and data base with the following commands:

```
PAB>DRIVER/-HD/-MM,DRIVER,DRIVER=
         ; 1) The /-HD switch is specified since a task header is not
         ;       needed. The driver is not a task but rather an
         ;       extension to the executive.
         ; 2) The switch /-MM must be used in the command line.
         ; 3) A map file is produced and is useful for debugging.
         ; 4) A symbol table file is specified since it will be
         ;       required by PROLOD.
PAB>DRVCOD
PAB>DRVTAB
```

```
  ; 5) PAB reprompts for input files. Both the code and the
  ;     data base are specified. The driver data base is built
  ;     into the image following the driver code.
PAB>[1,5]POS.STB/SS
PAB>[1,5]EXELIB/LB
PAB>/
  ; 6) POS.STB is specified as input so that executive routine and
  ;     listhead references may be resolved.
  ;     Note that /SS is required because the presence of certain
  ;     global symbols, such as $INTSV, cause PROLOD to reject
  ;     load attempts.
  ; 7) EXELIB is specified so that data structure offsets, mask,
  ;     and bit references may be resolved.
  ; 8) The single slash begins the option phase of the task builder.
Enter options:
TKB>STACK=0
  ; 9) The taskbuilder is directed not to allocate stack space. A
  ;     driver uses the executive's stack sparingly.
PAB>PAR=GEN:120000:40000
PAB>//
  ; 10)A partition, base virtual address and maximum size are
  ;     specified and the double slash terminates task builder
  ;     input.
```

5.1.4 Loading an I/O Driver Into the System

After completing the taskbuilding of your driver without any detected errors, you must create a small task to issue the POSSUM "PROLOD" system service call. If you are in the process of debugging the driver, it is suggested that you attach a debugging terminal to the printer port (TT2:), define the logical name "PROLOD$MSG", equating it to "0", and run XDT prior to invoking PROLOD.

5.2 PROLOD

The callable PROLOD system routine provides a method to load or unload an I/O driver into P/OS. PROLOD is an entry point in the POSSUM system resident library; the routine calls the server task $xxLOAD. (See the *P/OS System Reference Manual* for a general description of POSSUM system services.)

To avoid memory fragmentation, PROLOD loads the driver in the highest available physical memory, checkpointing any eligible regions that may be resident in high memory.

PROLOD requires that both the driver image (.TSK) and the driver symbol table (.STB) be located in LB:[ZZSYS]. PROLOD provides the primary values for the device, directory, filename extension, and last three letters (DRV) of the filename itself. You cannot specify these in the request, and the name of the device in D.NAM must be the same as the two characters provided. Also, PROLOD uses only the highest version of the files.

Unload operations currently have a restriction that requires access to the image and symbol table files so that PROLOD can easily determine which units and controllers need to be placed offline prior to unloading the driver code from memory. The driver's data base is never removed from primary pool and, if the driver is reloaded, it is assumed that the old data base can be reused.

Optionally, a driver can specify the $xxLOA and $xxUNL entry points so that it can save, restore, or initialize any required context.

To load or unload a driver, invoke PROLOD with the following arguments:

```
status, request, fnam, fnamsz
```

where:

status	The address of the 8-word Status Block. The chapter on POSSUM in the *P/OS System Reference Manual* describes the Status Block.
request	The address of a word containing a request value indicating the operation to be performed. The defined decimal values are: 1. = Load a driver and bring online all associated controllers and units. 2. = Unload a driver after successfully bringing all associated controllers and units offline.
fnam	If the request is to load or unload, this argument contains the address of a word containing two ASCII characters that PROLOD can use to derive the driver task filename and the symbol table filename. (Both files must be present to successfully load a driver.)

The derived names are of the form LB:[ZZSYS]xxDRV.TSK and LB:[ZZSYS]xxDRV.STB, respectively, where xx are the two characters you supply. For example, you supply the two characters BM for a driver named BMDRV.

fnamsz	If the request is to load or unload a driver, this argument contains the address of a value indicating the size in bytes of the filename argument. This value is currently 2.

5.2.1 PROLOD Operations and Diagnostic Checks

The PROLOD routines relocate and validate many of the pointers within the data base and, in the process, validate other data in the structures. The driver itself is then loaded into the highest available physical address space, and the data base is loaded into system pool.

To read the data base from the driver image file into the system pool, the global labels $xxDAT and $xxEND, defining the start and end of the data base, are needed.

To check the data base, the PROLOD routines must know the starting address of the DCB. If the global label $xxDCB is not defined (it is not in the symbol table file), PROLOD assumes that the DCB is the first word of the data base. When this assumption is incorrect because the DCB is elsewhere in the data base and is not labeled properly, many unusual error conditions result. To avoid this type of problem, you should always define the start of the DCB with the global label $xxDCB.

Each CTB is checked and relocated. The following offsets are both checked and relocated:

L.LNK	The link to the next CTB must be even. If it is not zero, it must point within the data base, and the CTB to which it points must lie within the data base. (Because it is highly unusual to have two controller types in one driver data base, this value is usually zero.)
L.DCB	The address of the related DCB must be even, point within the data base, and the DCB to which it points must lie within the data base.

	The DCB address(es) in the table must be even, and the DCB(s) to which each address points must lie within the data base.
L.DID	If non-zero, PROLOD scans the system configuration table for the presence of a device with a matching hardware ID. If a match is found, then PROLOD uses information in the system configuration table to assign the values K.SLT, K.ICSR, K.CSR and K.VCT of the KRB. See Section 5.2.2 for information about specifying L.DID during PROLOD autoconfiguration.
L.KRB	Each pointer in the table of KRB addresses must be even and must point within the data base, and the KRB to which each cell points must lie within the data base.

The following offsets in the CTB are checked:

L.NAM	The controller name cannot duplicate other L.NAM entries in the loadable data base.
L.NUM	The number of controllers, must be less than 17 (decimal).

Each KRB is checked and relocated. The following offsets in the KRB are both checked and relocated:

K.OWN	The pointer to the owner UCB must be even and point within the data base, or be zero. If it is nonzero, the pointer is relocated.
K.OFF	The start of the table of UCB addresses produced from K.OFF must be even and must point within the data base. The entries themselves must be even, point within the data base, and the UCB to which each cell points must lie within the data base.
K.CRQ	The listhead for the controller request queue.
K.CRQ+2	It is initialized to an empty list with the first word zero, and the second word pointing to the first, relocated.

Each DCB is checked and relocated. The following offsets are both checked and relocated:

D.LNK	The link to the next DCB must be even. If it is nonzero, it must point within the data base, and the DCB to which it points must lie within the data base.
D.UCB	The link to the first UCB must be even and must point within the data base, and the UCB to which it points must lie within the data base.
D.UCBL	The length of the UCB must be even and nonzero.
D.UNIT	The highest unit number (increased by 1) used with D.UCBL forms the last address of all UCBs. This address must lie within the data base.

The pointer to the driver dispatch table (D.DSP) is set to zero to show that the driver is not yet loaded.

Each UCB is checked and relocated. The following offsets are both checked and relocated:

U.DCB	The pointer to the DCB must point to the DCB that points to this UCB.
U.SCB	The pointer to the SCB must be even, must point within the data base, and the SCB to which it points must lie within the data base.
U.RED	The unit redirect pointer must be nonzero and even if it is an Executive address. If it is not an Executive address, it must be nonzero, even, and point within the data base.

Each SCB is checked and relocated. The following offsets are both checked and relocated:

S.KRB	The pointer to the KRB must be even, must point within the data base, and the KRB to which it points must lie within the data base. If S.KRB is nonzero, there must be a CTB in the loadable data base.
S.KTB	If the table of KRB addresses is present, each entry must point within the data base. (PROLOD preserves bit zero in each entry.) Each entry in the table must also have a matching entry in the table of KRB addresses

of a CTB in the loadable data base.

The following offsets in each SCB are initialized as described:

S.LHD	The head of the I/O queue is set to zero and the pointer to the end of the queue (S.LHD+2) is set to point at S.LHD.
S.PKT	The pointer to the current I/O packet is set to 1.

These last checks end the loading and validating of the data base.

After the data base is loaded and validated and no error is found, the driver itself is loaded into memory. In loading the driver, the driver dispatch table is validated, each interrupt entry in the driver dispatch table is inspected, and the vector(s) are checked. If a vector address is higher than the highest available vector address PROLOD prints a warning message. Interrupt control blocks are created and linked into the list starting at L.ICB in the CTB.

The format of the DDT must be consistent with that described in Section 4.3.1. If the device that the data base describes does not have any physical controllers (that is, a CTB does not exist), the DDT is not checked. Otherwise, the device has at least one interrupt vector and therefore at least one interrupt entry point. The DDT is then checked. The two global labels $xxTBL and $xxTBE must define the start and end of the DDT. The generic controller name(s) must be nonzero and the interrupt entry values must be valid. Interrupt entry point 0 must be nonzero, even, and lie in the range 117777 and 140000. If the format of DDT is inconsistent, PROLOD prints an error message, restores the system device tables, and exits.

When the driver is loaded, all links are established. The DCB of the loadable data base is put in the list of DCBs just in front of the DCB for the first pseudo device. The CTB(s) are linked to the end of the CTB list. The DDT address D.DSP, the driver PCB address D.PCB, and the driver mapping S.KS5 (the block number of the first word of the driver) in the fork block are initialized. The address of the start of the KRB table in the CTB, denoted in the driver data base by the local label $xxCTB, is loaded into the DDT.

5.2.2 Using PROLOD Autoconfiguration

PROLOD autoconfiguration provides option slot independence for your driver. When you use autoconfiguration, PROLOD automatically determines the option slot in which a particular device resides. Additionally, PROLOD sets the controller CSR, the vector, the interrupt controller CSR, and the slot number in the driver's data base.

We recommend that all drivers for P/OS systems use autoconfiguration because of the option slot independence it provides. This independence eliminates slot contention and simplifies end user installation.

To use autoconfiguration, specify a hardware option device ID in the L.DID field of the driver's CTB. The hardware option device ID identifies the controller type. Also, you must specify the value of K.CSR to be equal to zero.

During autoconfiguration, if PROLOD successfully finds the device and gains access to that device for the driver (by attempting to offline a driver which is currently using the device), then PROLOD sets the values of K.CSR, K.ICSR, K.SLT, and K.VCT.

As an alternative to using autoconfiguration, you can explicitly access a particular slot by preconfiguring your driver data base. Note that drivers that access system options such as the kernel Communications Port must always preconfigure their controller data base.

To preconfigure your driver data base, specify the following values:

- K.CSR--associated controller CSR
- K.ICSR--associated interrupt controller's CSR
- K.SLT--the slot number in which the option is present

PROLOD does not check the values K.ICSR and K.SLT. Once loaded, your driver can change K.ICSR or K.SLT for its own purposes; however, the value of K.CSR must always be within the address range of those CSRs assigned to the particular option. PROLOD derives the controller's vector address (K.VCT) from the specified CSR.

5.2.3 PROLOD Errors

PROLOD attempts to return error message text if and only if you have defined the logical name PROLOD$MSG with the equivalence value 0. Multiple errors may be detected in the course of loading and checking the driver and its data base.

PROLOD returns the following server-specific error codes in word 2 of the Status Control Block when word 1 contains the value -4 (server-specific error):

0.	Illegal request format
- 1.	File not a valid driver task image
- 2.	Privileged command
- 3.	Inconsistent argument length
- 4.	Illegal request function code
- 5.	Illegal unit name format specified
- 6.	Specified unit not found
- 7.	Illegal controller name specified
- 8.	Specified controller not found
- 9.	Failed to offline device
- 10.	I/O error on input file
- 11.	Failed to bring device online
- 12.	Controller already online
- 13.	Unit already online
- 14.	File has illegal STB format
- 15.	Device not found in system
- 16.	Illegal device name specified
- 17.	Data base for not found in system or driver
- 27.	Partition/region not in system
- 33.	Device not mounted
- 34.	File not contiguous
- 35.	Open failure on file
- 40.	Task image I/O error in file
- 46.	Partition too small
- 50.	Illegal driver task APR usage
- 51.	Partition/region is a common
- 60.	Driver already resident
- 61.	Driver being loaded or unloaded
- 62.	Insufficient pool space
- 63.	Loadable driver support not in system
- 64.	Driver not loaded
- 65.	Driver cannot be unloaded, still online
- 66.	Device is attached, busy, online and/or mounted
- 68.	Invalid driver data base at offset in file
- 69.	Driver built with wrong executive STB file
- 70.	Warning - KRB interrupt vector too high
- 72.	Warning - KRB interrupt vector in use
- 73.	Symbol is undefined in file
- 74.	Symbol is doubly defined by file
- 75.	Illegal value for symbol in file
- 76.	Driver dispatch table is inconsistent
- 77.	CTB is not supported by driver -- not loaded
- 78.	Cannot load/unload a pseudo device
- 79.	Too many symbols of the form in file
- 80.	CTB does not exist

- 81.	DCB table for CTB is full
- 82.	KRB table of CTB will not accept KRB
- 83.	KRB not in loadable data base
- 84.	CTB name is a duplicate
- 85.	Warning - loadable driver larger than 4K
- 86.	Partition/region is not a common
- 87.	KRB is not offline
- 88.	Illegal use of partition or region

5.2.4 PROLOD Sample Program

The following macro program illustrates a PROLOD call.

```
        .TITLE  LOAD - Example PROLOD call

        .mcall  qiow$s,exit$s           ;request system macros
;
; This task requests PROLOD to load the driver XXDRV from LB:[ZZSYS]
;
start:  mov     #loaarg,r5      ;get args for sample call to PROLOD
        call    prolod          ;load the driver
                                ;print status block always
        mov     #stat,r2        ;point to status args
        mov     #fmt,r1         ;output format
        mov     #outbf,r0       ;output buffer
        call    $edmsg          ;format it
        qiow$s  #io.wlb,#5,#5,,,,<#outbf,r1>    ;print it
        exit$s                  ;exit

;
; local data
;

loaarg: .word   4               ;four arguments (simple load call)
        .word   stat            ;pointer to eight word status block
        .word   rqst            ;pointer to PROLOD request
        .word   fnm             ;pointer to filename size
        .word   fnmsiz          ;pointer to size (in bytes) of fnm
stat:   .blkw   8.              ;eight word status block
fnmsiz: .word   fnmsz           ;word containing size of fnm
rqst:   .word   1               ;load (and online all) request
fnm:    .ascii  /XX/            ;driver name
                                ;implied filename "LB:[ZZSYS]XXDRV.TSK"
                                ; AND "LB:[ZZSYS]XXDRV.STB"
fnmsz=.-fnm                     ;name extension as this can't be spec'd)

outbf:  .blkb   132.            ;output buffer for status display
fmt:    .asciz  /%Nload request status: %N%8P/ ;output format string
```

```
.even

.end    start
```

CHAPTER 6

DEBUGGING A USER-SUPPLIED DRIVER

Adding a user-supplied driver carries with it the risk of introducing obscure bugs into a P/OS system. Because the driver runs as part of the Executive, P/OS provides an Executive debugging tool (XDT).

6.1 THE EXECUTIVE DEBUGGING TOOL

XDT is an interactive debugging tool which can aid in debugging Executive modules, I/O drivers, and interrupt service routines. This debugging aid is similar to ODT, the task-level debugger. XDT occupies physical address space in the GEN partition but does not take up any Executive virtual address space. XDT also does not interfere with user-level ODT, which can be used with any number of tasks while you are debugging your driver with XDT.

6.1.1 XDT Commands

XDT commands are generally compatible with ODT commands. XDT does **not** contain the following commands available in ODT:

- No $M - (Mask) register
- No $X - (Entry Flag) registers
- No $V - (SST vector) registers
- No $D - (I/O LUN) registers
- No $E - (SST data) registers
- No $W - (Directive status word) $DSW word

- No E - (Effective Address Search) command
- No F - (Fill Memory) command
- No N - (Not word search) command
- No V - (Restore SST vectors) command
- No W - (Memory word search) command

In addition, the X (Exit) command in XDT will simply issue a HALT instruction. This will drop the printer port terminal into Micro-ODT. (See Section 6.2.

6.1.2 XDT Start Up

You may install XDT from DCL with the command: INSTALL XDT/NOREMOVE. The "noremove" switch will inform the executive not to abort XDT when the DCL or PRO/Tool Kit application exits. From DCL, use the SPAWN RUN XDT command. An initialization message will appear. When active, XDT runs entirely at priority level 7.

6.1.3 XDT General Operation

Prior to embarking on an XDT debugging session, plug a maintenance cable (BCC-08) into the printer port and attach a VT100 or similar terminal. A maintenance cable is the same as a printer cable (BCC-05), except that pins 8 and 9 are shorted. If the cable was in place when the system was turned on, the terminal must be set to 9600 baud; otherwise, the terminal must be at 4800 baud. Input and output are directed to this port.

Assemble the driver with an embedded BPT instruction, or use the ZAP utility to set the breakpoint by replacing a word of code with the BPT instruction. (Make sure to write down the instruction that you replace with the BPT instruction.) When the breakpoint instruction in the driver is executed, XDT prints:

```
BE:xxxxxx
XDT>
```

Then:

1. Using XDT, replace the BPT instruction with the desired instruction.

2. Decrement the PC by subtracting 2 from the contents of register R7.

3. Then proceed by using the P or S commands, after optionally setting breakpoints within the driver or examining memory locations.

6.1.4 XDT and Debugging a User-Supplied Driver

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

```
TE:xxxxxx
XDT>
```

This error results when control reaches a breakpoint that you have set, using XDT, in a loaded driver. The T-bit error, rather than the expected BE: error, occurs unless register APR5 is mapped to the driver at the time XDT sets the breakpoint. Assembling or zapping the BPT (as opposed to setting it from an XDT prompt) will help avoid this T-bit error.

NOTE

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

6.2 MAINTENANCE- OR MICRO-ODT

The processor microcode supports a more limited set of debugging commands which permit debugging of a system in an otherwise inaccessible state. Be advised, however, that Micro-ODT is able to access only the low 28K words of memory and the I/O Page, whereas XDT can be mapped to any part of memory. Micro-ODT is more fully documented in the *Professional 300 Series Technical Manual*. It will also be discussed further in this chapter.

6.3 FAULT ISOLATION

Four causes can be identified when the system faults:

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

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

6.3.1 Immediate Servicing

Faults manifest themselves in four ways as listed below (in order of increasing difficulty to isolate):

1. If XDT is running, an unintended trap to XDT occurs.
2. The system displays a software bugcheck and halts.
3. The system halts but displays nothing.
4. The system is in an unintended loop.

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

6.3.1.1 The System Traps to XDT - A trap may or may not be intended (for example, a previously set breakpoint). If it is not intended and you have some idea of the source of the problem (for example, a recent coding change), you may use XDT to examine pertinent data structures and code.

6.3.1.2 The System Halts but Displays No Information - Before taking any action, preserve the current PS and PC and the pertinent device registers (that is, examine and record the information these registers contain). See Section 6.2.

6.3.1.3 The System Is in an Unintended Loop - Proceed as follows:

1. Halt the processor by pressing <BREAK> from the debugging terminal (Micro-ODT).
2. Record the PC, the PS, and any pertinent device registers, as in Section 6.3.1.2.

You may then want to step through a number of instructions in an attempt to locate the loop. Toggle the Micro-ODT Halt register by entering "H" at the "@" prompt. Following this, each "P" command will result in a single step. To restore "proceed" functionality to the "P" command, enter "H" again.

In order to avoid the side-effects of printer port interrupts you may first wish to disable printer receiver interrupts. Then proceed as follows:

1. In micro-ODT open location 173202 (the CSR).
2. Set the printer receiver IMR (interrupt mask register) bit by depositing a 75 in this location.

6.3.2 Pertinent Fault Isolation Data

Before you attempt to locate the fault, you should examine the system common (SYSCM). SYSCM contains a number of critical pointers and listheads. In addition, you should examine the dynamic storage region (system pool and ICB pool) and the device tables. The device tables are in the module SYSTB. At this point, you have the following data, which represents a minimal requirement for effectively tracing the fault:

- PS
- PC
- The stack
- R0 through R6
- Pertinent device registers
- The dynamic storage region

- The device tables
- System common

6.4 TRACING FAULTS

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

$STKDP - Stack Depth Indicator

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

+1 -- User (task-state) stack or a privileged task at user state

0 or less -- System stack

If the stack depth is +1, then the user has crashed 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 (Pool-Resident)

Figure 6-1 shows the interaction of task header pointers. Note that all tasks on P/OS systems have a pool-resident (external) header.

The pointers $HEADR, $SAHPT, and $SAVSP all point to the first word of the task header, which contains the user task's stack pointer (SP) from the last time it was saved.

Figure 6-2 shows a brief description of the task header. The task header is fully described in the *RSX11M/M-PLUS and Micro/RSX Task Builder Manual*, which is distributed with the Tool Kit Documentation.

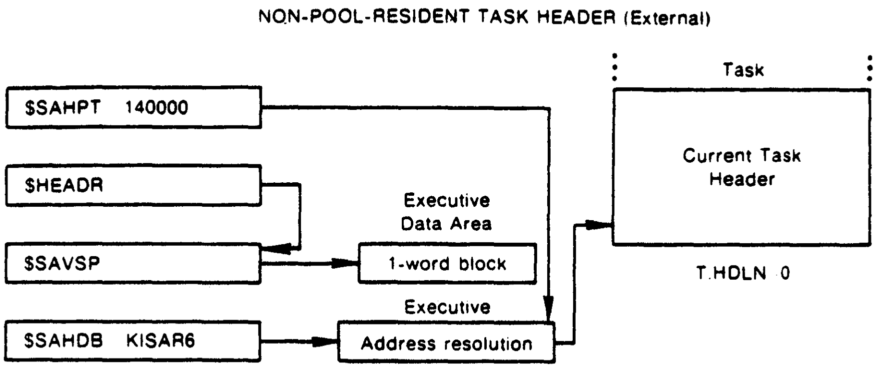


Figure 6-1: Interaction of Task Header Pointers

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

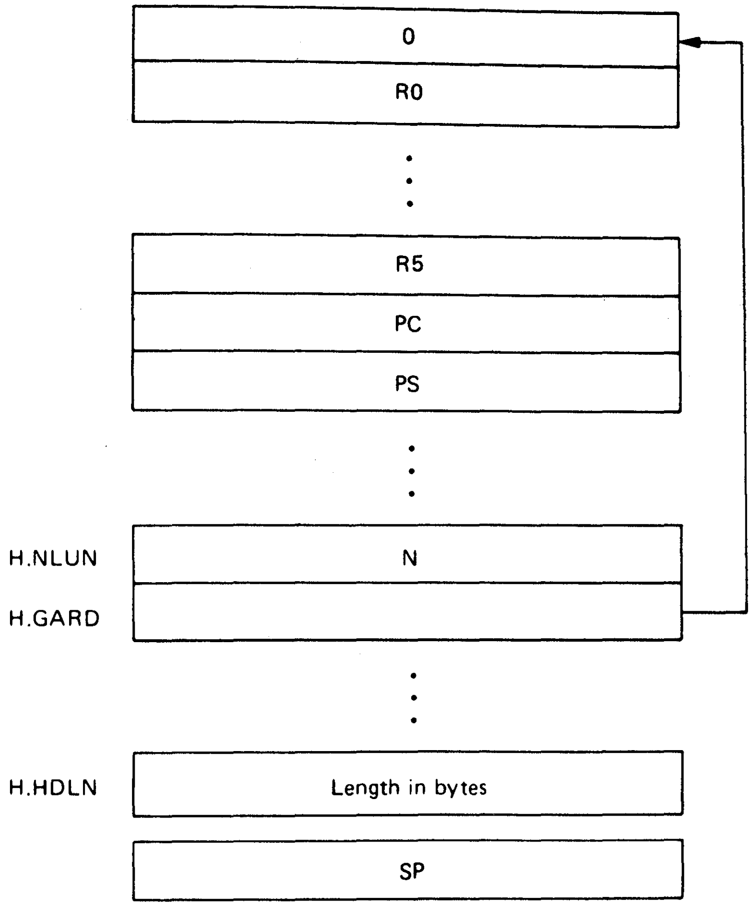


Figure 6-2: Task Header

The pointers associated with a pool-resident (external) header are described next:

$HEADR - Points to the current task header.

The $HEADR word points to the pool-resident task header of the task currently running. The value in $HEADR is a kernel virtual address in primary pool.

$SAVSP - Points to the first word of the current task header, which contains the saved stack pointer.

$SAHPT - Points to the current task header in pool. $SAHPT contains the virtual address of the header. $SAHPT and $HEADR contain the same virtual address for a pool-resident header.

$SAHDB - Contents undefined .x $SAHDB

6.4.1 Tracing Faults Using the Executive Stack and Register Dump

The following discussion implies that XDT is active. If the system crashes while XDT is not active, a software bugcheck occurs. Procedures for analyzing a bugcheck are discussed in Section 6.4.5. To trace a fault after a software crash, first examine the system stack pointer. Usually an Executive failure is the result of an SST-type trap within the Executive. If an SST does occur within the Executive, then the origin of the call on the crash-reporting routine is in the SST service module. (The crash call is initiated by issuing an IOT at a stack depth of zero or less.)

A call to crash also occurs in the Directive Dispatcher when an EMT is issued at a stack depth of zero or less, or a trap instruction is executed at a stack depth of less than zero. The stack structure in the case of an internal SST fault is shown in Figure 6-3.

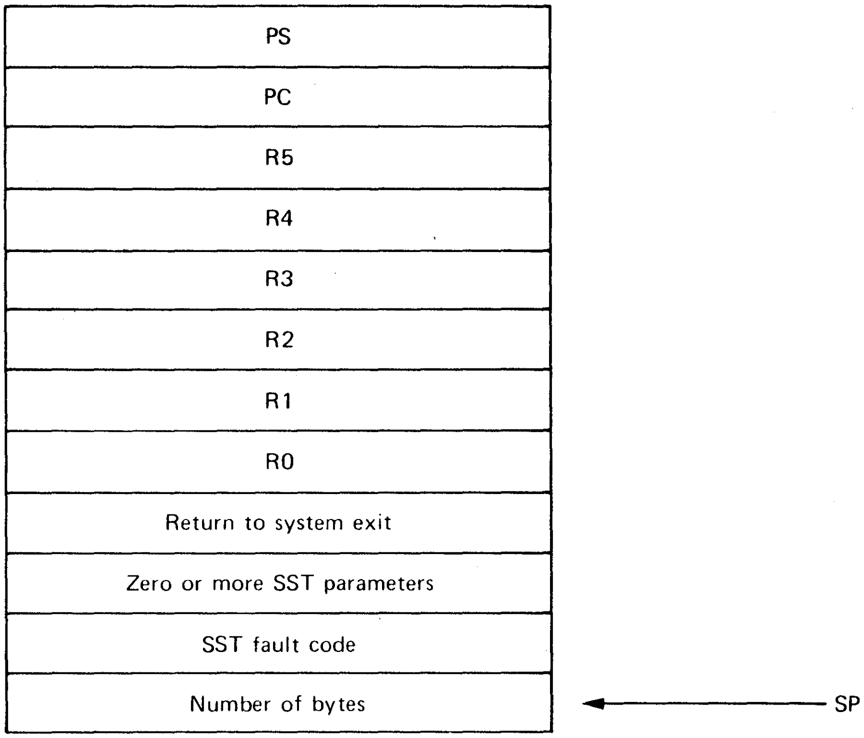


Figure 6-3: Stack Structure: Internal SST Fault

The fault codes are:

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

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

If the failure is detected in $DRDSP, the stack is the same as that shown in Figure 6-3, except that the number of bytes, the SST fault code (the fault codes are listed above), and the SST parameters are not present.

One SST-type failure, stack underflow, does not result in the stack structure of Figure 6-3. To determine where the crash occurred, first establish the stack structure. This can be deduced by the value of the SP and the contents of the top word on the stack. If the stack structure is that of Figure 6-3, then the failure occurred in $DRDSP, or was a normal SST crash. If the stack structure is that of Figure 6-4, then an abnormal SST crash has occurred.

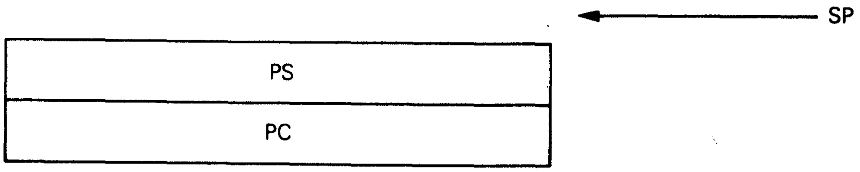


Figure 6-4: Stack Structure: Abnormal SST Fault

Abnormal SST failures occur when it is not possible to push information on the stack without forcing another SST fault. When this situation occurs, a direct jump to the crash-reporting routine is made rather than an IOT crash. The PS and PC on the stack are those of the actual crash, and the address printed out by the crash-reporting routine is the address of the fault rather than the address of the IOT that crashes the system. Note that the crash-reporting routine removes the PC and PS of the IOT instruction from the stack, which in this case is incorrect. Thus, the SP appears to be four bytes greater than it really is (as in Figure 6-4).

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

6.4.2 Tracing Faults When the Processor Halts Without Display

To trace a fault when the processor halts but displays no information, first examine $STKDP, $TKTCB, $HEADR, $SAVSP, $SAHPT and $SAHDB. The difficulty in tracing failures in this case is that the system stack is not directly associated with the cause of a failure.

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

Frequently, a fault can occur that causes the SP to point to Top of Stack (TOS)+4. This fault results from issuing an RTI when the top two items on the stack are data. The result is a wild branch and then, most probably, a halt. Figure 6-5 shows a case in which two data items are on the stack when the program executes an RTI. TOS points to a word containing 40100. Suppose that location 40100 contains a halt. This indicates that the original SP was four bytes below the final SP, and fault tracing should begin from the original SP.

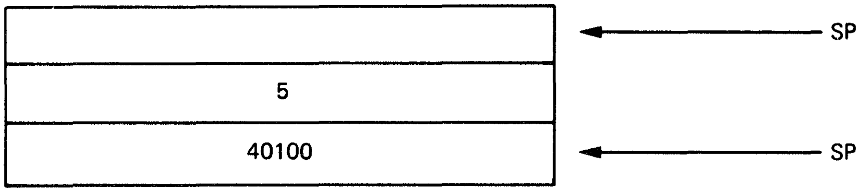


Figure 6-5: Stack Structure: Data Items on Stack

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

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

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

6.4.3 Tracing Faults After an Unintended Loop

To trace a fault when an unintended loop has occurred, first halt the processor by hitting <BREAK> from the debugging terminal.

After you halt the processor, the same state exists as was discussed in Section 6.4.2. Follow a similar tracing procedure to the one described there. A specific suggestion is to check for a stack overflow loop. Patterns of data successively duplicated on the stack indicate a stack looping failure.

6.4.4 Additional Hints for Tracing Faults

Another item to check is the current (or last) I/O Packet, the address of which is found in S.PKT of the SCB. The packet function (I.FCN) defines the last activity performed on the unit.

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

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

6.4.5 System Bugcheck Without XDT

If a software error causes the system to crash while XDT is not active, the system will bugcheck. A bugcheck will request the diagnostic ROM to display an unhighlighted picture of the Professional with a two-number code, for facility and type of bugcheck. Errors with a facility code of 000300 are executive or other system state errors, in which case, the type is represented in the following list:

000000 IOT in System State

TRACING FAULTS

000001 Stack Overflow

000002 Trace Trap or Breakpoint

000003 Illegal Instruction Trap

000004 Odd Address or Other Trap 4

NOTE

> Since there are no odd address traps or memory parity errors on the 350, these traps are most likely NXM (non-existent memory), caused by illegal address computation, bus timeout, or memory management fault.

000005 Segment Fault

000006 A Background Task (one without a parent) aborted or exited with I/O outstanding

The processing of the diagnostic ROM changes some of the context of the system at the time of the crash, so that the diagnosis of the problem is made somewhat more difficult. In particular, after a bugcheck, KISAR5, KISAR6, UISAR0, and UISAR1 have been altered, and the Kernel Stack Pointer has not been preserved. You must therefore examine the Kernel stack from $STACK toward low memory in order to locate the trap.

When the processor traps, the PSW and PC are pushed onto the stack. If the trap is a directive issued by a task in user state (i.e. an EMT trap), the task's general purpose registers are pushed beginning with R5 and ending with R0. Then the return call to $DIRXT and an initial DSW success code of 1 are pushed onto the stack. The directive processing may call other system subroutines, some of which might save R5 and R4 (by convention the "nonvolatile" registers). When the stack contents are analyzed to determine whether they are pointing to data structures or subroutine addresses, a picture of the flow of control can be outlined. Often there will be pointers on the stack to UCB's or other driver-related structures, TCB's or other task-related structures, or saved APR mapping biases for temporary remapping of KISAR5 and KISAR6. Since most of these structures are in primary pool (and thus in the low 28KW of physical memory), they may be examined using micro-ODT.

Ultimately, a crash will be preceded by a trap in system state. Again, the PSW and PC at the time of the crash will be pushed onto the stack. A PSW of 0300nn or 000nnn indicates Kernel mode, previous mode User or Kernel, respectively. The PC will point to the instruction following the one causing the trap. Remember that after a bugcheck R6 (or SP) will not point to this address on the stack.

6.5 REBUILDING AND REINCORPORATING A DRIVER

After correcting and assembling the driver source, unload the old version using PROLOD, task build the new one, and load it using PROLOD.

Once loaded, the data base is not removed by PROLOD. If the data base is incorrect and cannot be patched, correct its source, reassemble it, and build the new driver task. Then bootstrap the system before loading the driver task image containing the corrected data base.

CHAPTER 7

EXECUTIVE SERVICES AVAILABLE TO AN I/O DRIVER

Because a driver is mapped within the Executive address space, it can call Executive routines on the same basis as that of any other module in the Executive. The driver must observe the protocol and conventions established on the system. The following sections summarize the conventions, describe the address double word, tell what special processing is required for NPR devices attached to a PDP-11 processor with extended memory support (22-bit addressing), and summarize some of the typical Executive services available.

7.1 SYSTEM-STATE REGISTER CONVENTIONS

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

When a driver is entered directly from an interrupt, it is operating at interrupt level, not at system state. At interrupt level, any register the driver uses must be saved and restored. INTSV$ generates code to preserve R5 and R4 for the driver's use. All drivers must follow these conventions.

See the description of the driver dispatch table in Section 4.5 for the contents of registers when a driver is entered.

7.2 EXECUTIVE TIMER RELATED FACILITIES

The executive provides a number of timer related facilities which are available to an I/O driver. They are typically used to periodically poll a device for a status change in the absence of an interrupt capability, to provide general timer services to the driver so that it can perform it's own timeout processing (needed by more advanced drivers such as the full duplex terminal driver), or to call the driver back after some specified period of time.

A timer request requires a structure called a clock block. It is normally located in the device driver's database which has been allocated from primary pool and is "C.LGTH" bytes in length. A driver can however simply allocate the clock block from primary pool using the $ALOCB executive subroutine. The clock block contains the request type, the absolute time when request is to occur, and the bias and APR5 displacement of the driver routine to be called when inserted in the system clock queue. A timer request is made by calling the executive's $CLINS routine. Input parameters to $CLINS are the virtual address of the primary pool resident clock block, the request type of "C.SYST", the high and low order time delta (in tics), and a unique identifier. "C.SUB" is expected to contain the virtual address of the driver routine to be called by executive's clock processor (executive module TDSCH) when the specified interval of time has elapsed. The unique identifier is used to dequeue timer requests before completion and must be a unique executive (primary pool) virtual address. This unique identifier could be a UCB address, or even the address of the clock block itself. Note that since this identifier is a system wide identifier, and an ad hoc value could potentially corrupt the system.

Timer requests are one shot in nature and as such, the clock block is dequeued by the TDSCH processing. The request must be respecified via the $CLINS routine to achieve a periodic timer facility. The driver subroutine specified in the request will be called at system state at processor priority 0.

If it is necessary to cancel a timer request, this may be done by removing the clock block from the system's clock queue via the executive's $CLRMV routine. The following examples further illustrate the manipulations required:

```
;+
; Example clock queue insertion and removal
;
; Driver database contains clock block in UCB and thus avoids resource
; allocation errors associated with dynamic allocation from primary
; pool.
;
;        |                |
;        +----------------+
;        | .BLKB C.LGTH   | <--- driver specific clock block at U.MYCB
;        +----------------+
;        |                |
;
;             UCB
;
;
```

```
; examples assume R5 => UCB
;
;-

; The first example is a periodic polling of some device status. Though
; status changes are traditionally communicated to the driver via an
; interrupt, there are instances of when this hardware functionality is
; not available.

INITIM: MOV     R5,R0              ;copy UCB address
        ADD     #U.MYCB,R0         ;point to clock block
        MOV     #POLL,C.SUB(R0)    ;specify address of timer routine
        CLR     R1                 ;zero high order delta time
        MOV     #H$$RTZ/2,R2       ;get number of ticks per half
                                   ;second
                                   ;for low order time delta
;
; Note that for a request type (6) "C.SYST" $CLINS will save
; current APR5 mapping and restore it when time delta expires
; before calling driver. The implication is that the virtual
; address specified in C.SUB must be mapped through APR5 and that
; the caller of $CLINS is also mapped through APR5 with the same
; mapping. If a request type (8) "C.SYTK" is specified, then the
; caller must have already specified the APR bias in C.AR5 in
; addition to the virtual address in C.SUB. Calling $CLINS after
; the clock block has already been inserted in the clock block can
; cause unpredictable results including the removal of other system
; timer requests as a result of the clock queue corruption. Should
; this be a problem, C.RQT could be used as an interlock by
; clearing it after clock queue removal and checking this word to
; determine if the clock block is in use before attempting to call
; $CLINS.
;

        MOV     #C.SYST,R4         ;specify request type
        CALLR   $CLINS             ;insert clock block into clock
                                   ;queue
                                   ;and return to this subroutine's
                                   ;caller.
;
; The driver is called at this entry point every .5 seconds. All
; registers may be used by the driver and upon entry, R4 contains
; the address of the clock block dequeued.
;
POLL:                              ;first, respecify clock request
        MOV     R4,R0              ;specify address of clock block
        CLR     R1                 ;init high order delta time
        MOV     #H$$RTZ/2,R2       ;init low order delta time
        MOV     C.TCB(R0),R5       ;get address of identifier (UCB
                                   ;address despite symbolic offset
                                   ;name)
```

```
                                 ;note that C.SUB is not modified
                                 ;and still valid.
        CALL    $CLINS           ;respecify timer request
                                 ;note that C.AR5, or C.SUB need
                                 ;not be respecified.
;
; On return from $CLINS, (R0,R4, and R5) are unmodified.
        .
        .                        ;driver does what needs to be
        .                        ;done
        .

        RETURN

;
; The second example builds on the first example and illustrates
; how to cancel a timer request.
;
; The CANTIM routine may be called to remove the clock block
; inserted above.
;

CANTIM: MOV     R5,R1            ;specify identifier of clock
                                 ;block
                                 ;(in this case, the UCB address)
;
; A clock block may be removed from the system clock queue however,
; the listhead $CLKHD is not a double word listhead and therefore,
; $QRMVT may not be called. $CLRMV can be called to remove clock
; blocks from the system clock queue and if the request type was
; not "C.SYST", the clock block will be deallocated from primary
; pool. Therefore, if the driver's clock block is located in the
; driver's database (such as in the UCB or KRB), only the request
; type "C.SYST" should be specified.
;

        MOV     #C.SYST,R4       ;specify type of request to
                                 ;dequeue
        CALL    $CLRMV           ;dequeue it (if there)
            .
            .

        RETURN                   ;return to caller
```

7.3 ADDRESSING A TASK BUFFER

A typical user task has no knowledge of the physical locations of every region mapped into its virtual space, since the physical addresses of a task's regions can change due to checkpointing or shuffling.* Given that a task can only specify the virtual address of a buffer to a system directive such as QIO$, this virtual address must be resolved to some form of an actual physical address being used while the issuing task's context is loaded. Most I/O operations are asynchronous to the execution of user tasks. As a result, the issuer could change its virtual address space mapping, and thus invalidate the specified virtual address. The executive assists the driver by keeping a count of all active I/O requests on a per region basis, and converting the user buffer virtual address in the first parameter to a physical address for transfer functions. If a region contains a non-zero I/O count, the executive will not checkpoint or shuffle the region. This would both invalidate the physical address and compromise the integrity of the system, since the outstanding I/O could be transferred into the another region resident in memory.

For non-DMA devices, there are three common methods for a driver to reference a task buffer:

1. The simplest approach is to move the "bias" half of the physical address double word into KISAR6 and then use the "displacement-in-block" half, which has been adjusted by the QIO directive processor to map through APR6 for transfer functions. Unless the bias and displacement are adjusted by the driver, the maximum size of the user task buffer is limited to <4096.-32.> words.

2. In certain cases, it is not possible for the driver to unmap KISAR6 because another structure, such as an intermediate buffer (or possibly the driver code itself), is required to be mapped. These cases can use a method in which the executive $BLXIO routine is used to temporarily unmap the driver in KISAR5, map the source and destination buffers through KISAR5 and KISAR6. It then performs the transfer and restores the mapping.

3. Another method is useful for drivers that sequentially empty or fill a task buffer word by word, or byte by byte. With this method, the executive's $GTBYT, $GTWRD, $PTBYT and $PTWRD routines unmap KISAR6, perform the transfer, adjust

---

* Checkpointing is the process of copying a region to a disk so it can be used by another contending region. Shuffling is the process of moving a region from one physical location to another location in order to reduce fragmentation. Though a fixed region or a non-checkpointable region cannot be checkpointed, it can be shuffled - provided the region has a zero I/O count and is not explicitly marked as non-shufflable (PS.NSF).

the bias and displacement of the target buffer. The buffer could be as large as 32KW and restore the KISAR6 mapping (see module BFCTL).

7.3.1 Address Checking a Task Buffer

Address checking is the process of validating that a buffer is fully mapped within the task's virtual address space, that a buffer is contained within one region, and that the issuing task has write access to the region. A buffer can be checked for read only access or read/write access, depending upon the I/O function. For instance, an IO.WLB or IO.WVB check the buffer for read-only access, since it is known that the transfer will not write to the region. Normally, a driver does not need to explicitly address check the I/O buffer, as the executive's QIO$ directive (see DRQIO) assumes this function for transfer and ACP I/O functions.

7.3.2 QIO Directive Processing Specifics

As the Executive processes a QIO directive, it checks for certain conditions and performs actions as follows:

- The specified LUN must be valid and assigned. If it is not, a directive error IE.ILU or IE.ULN is returned.
- The UCB address in LUN is resolved to the target UCB by following the redirect pointer U.RED.
- If I/O is stalled for the unit (US.SIO=1) or if a file operation is pending (bit 0 in the second LUN word =1), the issuing task's PC is modified so that the directive is reissued at a later point in time after a significant event occurs. The Files-11 reverification task (VER...) is allowed to break through a stalled I/O state.
- The driver must be resident. If not, a directive error (IE.HWR) is returned.
- If an optional event flag is specified, it is validated and cleared. If an invalid (non-existent) event flag is specified, a directive error (IE.IEF) is returned.
- An I/O packet is allocated from the primary pool. If there is insufficient pool to create the I/O packet, a directive error (IE.UPN) is returned.

- o If the directive QIOW$ is received, the task is placed in a "waitfor" state - if the event flag was specified.

- o An I/O rundown count (T.IOC) is incremented in the issuer's task control block.

- o The optional I/O status block is validated, and the contents cleared. The virtual address and physical address double word of the I/O status block are stored in the I/O packet.

- o I/O packet fields are initialized and used as indicated in Table 7-2.

- o IO.KIL functions are always legal and are processed immediately by the executive. The driver's I/O packet queue (listhead S.LHD) is flushed, whether the unit is online or not, of all packets with the same TCB address and UCB address if the device is not mounted with an associated ACP. If the unit is online and cancel notification has been requested by the driver (UC.KIL=1), the driver is always called at the D.VCAN entry point - independent of whether the unit was busy or if any packets were flushed. If the unit was busy at the time of the cancel I/O request, the driver will be called.

- o Depending on device characteristics, unit status, and type of function, specific processing is performed and the I/O packet is either queued to the driver or to the ACP.

Table 7-1: QIO Processing By Function Type and Device Characteristics

DV.MNT	TYPE	US.MNT	US.FOR	US.LAB	Checks and action performed
0	N	-	-	-	1,2,IS.SUC is returned
0	C	-	-	-	1,2,A
0	T	-	-	-	1,2,B
0	A	-	-	-	1,2,3,B
1	N	-	-	-	2,IS.SUC is returned
1	C	1	-	-	2,1,A,6
1	T	1	-	-	2,4,B,6
1	A	1	-	-	2,5
1	C	0	1	-	2,1,A,6
1	C	0	0	1	2,7,A,6
1	C	0	0	0	2,1,A,6
1	T	0	1	-	2,8,B,6
1	T	0	0	-	2,9,B,6
1	A	0	1	-	2,8,C,6
1	A	0	0	-	(beyond scope of this table)

where

N = NOP
C = Control
T = Transfer
A = ACP

1. If the device unit is allocated (U.OWN <> 0), and not public (US.PUB=0), then an I/O error of IE.PRI is return unless the issuing task is privileged or the issuing task's TI is the allocator ((T.UCB)=(U.OWN)).

2. If the function is not legal then an I/O error of IE.IFN is returned. Also, if the device unit is marked offline (US.OFL=1) then an I/O error of IE.OFL is returned.

3. If function code is IO.RVB ((I.FCN+1)=IO.RVB/^O400) then function code is converted to IO.RLB and all subfunction bits cleared. If function code is an IO.WVB, then subfunction is converted to an IO.WLB again clearing all subfunction bits.

4. If device unit's volume status is not valid (US.VV=0) an I/O error of IE.PRI is returned.

5. ACP functions to a dismounted volume result in an IE.PRI error.

6. Task load overlay checks. IO.LOV and IO.LOD have a particular meaning for mountable devices, whether or not device is mounted or not, and check is performed for both control and transfer functions. (These I/O functions are equivilent to IO.RLB!10 or IO.RLB!110 .)

7. Control functions to a mounted ANSI tape required the issuer to be privileged.

8. An ACP must be associated with the unit otherwise an I/O error of IE.PRI is returned.

9. A transfer function to a mounted volume must be a load overlay function or the task must be privileged.

    A. A control function format I/O packet is built and queued to the driver.

    B. A transfer function format I/O packet is built and queued to the driver. If the I/O function was an IO.WVB or an IO.WLB, then the buffer is address checked for read only access, otherwise the buffer is checked to ensure that the task has write access to the buffer's region.

C. Same action as B except that packet is queued to ACP.

Table 7-2: I/O Packet Usage by Function Type

I/O packet Field	Control Functions	Transfer Functions	Notes
I.LNK	utility link word		
I.PRI	(T.PRI)	(T.PRI)	
I.EFN	(Q.IOLU)	(Q.IOLU)	
I.TCB	($TKTCB)	($TKTCB)	1
I.LN2	($HEADR)+H.NLUN+2+<<Q.IOLU-1>*4>+2		2
I.UCB	redirected UCB address		3
I.FCN	(Q.IOFN)	(Q.IOFN)	4
I.IOSB	virtual address of I/O status block		5
I.IOSB+2	bias of I/O status block		5
I.IOSB+4	disp. in blck+140000 of I/O status blck		5
I.AST	virtual address of I/O completion AST		
I.PRM	(Q.IOPL)	bias of buffer	
I.PRM+2	(Q.IOPL+2)	DIB+140000 of buf.	
I.PRM+4	(Q.IOPL+4)	(Q.IOPL+2)	
I.PRM+6	(Q.IOPL+6)	(Q.IOPL+4)	
I.PRM+10	(Q.IOPL+10)	(Q.IOPL+6)	
I.PRM+12	(Q.IOPL+12)	(Q.IOPL+10)	
I.PRM+14	0	(Q.IOPL+12)	
I.PRM+16	0	0	
I.AADA	0	address of ADB	8

ADDRESSING A TASK BUFFER

I.AADA+2	0	0

Notes:

1. If the high bit in the event flag byte field is set, it indicates a virtual I/O function. I.PRM+16 is then treated as a FILES-11 lock block during I/O completion. Mass storage device drivers should ensure that I.PRM+16 is not used as temporary storage of I/O context.

2. If the bit 0 is set in the contents (not the address) of the second LUN word I.LN2, it indicates that this word contains the window block pointer. If the function is virtual and this word is even, the task's header (task region) has been locked in memory by incrementing the attachment descriptor I/O count and the task region's PCB I/O count.

3. UCB addresses in the task header are not fully resolved to the target UCB, so a level of indirection is possible. For instance, a task which is preinstalled in the system before boot may need to read a disk overlay from LB:. The UCB address of the pseudo device LB: is in the LUN, and the system has redirected the pseudo device to the physical boot unit during the boot process to DW1:, DZ1:, or DZ2:.

4. The I.FCN word consists of two bytes. The high byte is the function code and the low byte is the subfunction code. The subfunction consists of eight independent bits which are interpreted within the context of the function code. The exceptions are as follows:

   - The subfunction IQ.UMD (4) stamps any I/O function as a diagnostic function, independent of any device characteristics.

   - The subfunction IQ.X (1) means inhibit reties. It is of primary interest to mass storage device drivers.

   - The subfunction IQ.LCK (^O200) is used by FILES-11 ACP functions and will not be seen normally by the mass storage device driver.

   - Another subfunction bit of concern to mass storage device drivers occurs when "write checking" is requested on a mounted volume. This is indicated by an IO.WLB function with subfunction bit (20) specified.

5. I/O counts are not maintained for the I/O status block. Therefore, the I/O status block may not be accessed by the driver except at predriver initialization time (UC.QUE=1). I/O completion processing uses the physical address of the I/O status block, provided no event has occurred while I/O was outstanding that would have invalidated this address (such as unmapping a region, an EXTK$, or a checkpoint), as indicated by T3.MPC. If such an event has occurred, the I/O packet is converted into a kernel AST packet such that when the task's context is loaded, the virtual address can be used. Note that system's integrity is preserved here, rather than the issuing task's, if the task remapped or unmapped the I/O status block. In general, a task may not change the virtual mapping of it's I/O status block, but can unmap an I/O buffer if needed.

   If I.IOSB+4 is odd, it indicates that the I/O packet is internal I/O which was issued from another driver or system process. When I/O is completed, a specified kernel mode completion routine is called - rather than performing normal I/O completion processing (see $IOFIN in IOSUB for details). This is intended to be transparent to the driver.

7.4 THE ADDRESS DOUBLE WORD

P/OS can accommodate configurations whose maximum physical memory is 2048K words. Individual tasks, however, are limited to 32K words. The addressing is accomplished by using virtual addresses and memory mapping hardware. I/O transfers, however, use physical addresses 18 bits in length. Since the PDP-11 word size is 16 bits, some scheme is necessary to represent an address internally until it is actually used in an I/O operation. The choice was made to encode two words as the internal representation of a physical address and to transform virtual addresses for I/O operations into the internal doubleword format.

On receipt of a QIO directive, the buffer address in the Directive Parameter Block, which contains a task virtual address, is converted to address doubleword format.

The virtual address in the DPB is structured as follows:

Bits 0 through 5	Displacement in terms of 32-word blocks
Bits 6 through 12	Block number
Bits 13 through 15	Page Address Register Number (PAR_#)

The internal P/OS translation restructures this virtual address into an address doubleword as described in the following paragraphs.

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

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

For nonprocessor request (NPR) devices, the driver requirements for manipulating the address doubleword are direct and are discussed with the description of U.BUF in Section 4.4.4.

7.5 SERVICE CALLS

This section contains general commentary on the Executive routines typically used by I/O drivers. The descriptions of the routines are taken from the source code of modules linked to form the Executive. Table 7-3 summarizes the routines described in this section. Only the most widely used routines are described; however, many other Executive services are available. The source code for the related routines is in the MACRO-11 source files for the Executive modules.

Table 7-3: Summary of Executive Service Calls for Drivers

Routine Name	Location in Module	Function
$ACHKB	EXSUB	Address check for byte-aligned buffers
$ACHCK	EXSUB	Address check for word-aligned buffers
$ALOCB	CORAL	Alocate core buffer
$BLKCK	MDSUB	Check logical block number
$BLKC1	MDSUB	Check logical block number
$BLKC2	MDSUB	Check logical block number
$BLXIO	BFCTL	Move block of data
$CKBFI	EXESB	Check I/O buffer
$CKBFR	EXESB	Check I/O buffer
$CKBFW	EXESB	Check I/O buffer
$CKBFB	EXESB	Check I/O buffer
$CLINS	QUEUE	Clock queue insertion
$CVLBN	MDSUB	Convert logical block number
$DEACB	CORAL	Deallocate core buffer
$FORK	SYSXT	Create a fork process
$FORK1	SYSXT	Fork but bypass clearing timeout count
$GTBYT	BFCTL	Get byte
$GTPKT	IOSUB	Get an I/O packet
$GSPKT	IOSUB	Get a special I/O packet
$GTWRD	BFCTL	Get word
$INIBF	IOSUB	Initiate I/O buffering
$INTXT	SYSXT	Interrupt exit
$IOALT	IOSUB	Alternate entry to $IODON
$IODON	IOSUB	I/O done for completing an I/O request
$IOFIN	IOSUB	I/O finish for special I/O completion
$PTBYT	BFCTL	Put byte
$PTWRD	BFCTL	Put word
$QINSP	QUEUE	Queue insertion by priority
$RELOC	MEMAP	Relocate address
$REQUE	IOSUB	Queue kernel AST to task
$REQU1	IOSUB	Queue kernel AST to task
$TSPAR	REQSB	Test if partition memory resident for kernel AST
$TSTBF	IOSUB	Test for I/O buffering

$ACHKB
$ACHCK

7.5.1 Address Check

These routines are in the file IOSUB. A driver can call either routine to address-check a task buffer while the task is the current task. The Address Check routines are normally used only by drivers setting UC.QUE in U.CTL. See Section 8.1 for an example.

Calling Sequences:

```
     CALL      $ACHKB
```

or

```
     CALL      $ACHCK
```

Description:

```
; +
; **-$ACHKB-ADDRESS CHECK BYTE ALIGNED
; **-$ACHCK-ADDRESS CHECK WORD ALIGNED
;
; THIS ROUTINE IS CALLED TO ADDRESS CHECK A BLOCK OF MEMORY TO SEE
; WHETHER IT LIES WITHIN THE ADDRESS SPACE OF THE CURRENT TASK.
;
; INPUTS:
;
;       R0=STARTING ADDRESS OF THE BLOCK TO BE CHECKED.
;       R1=LENGTH OF THE BLOCK TO BE CHECKED IN BYTES.
;
; OUTPUTS :
;
;       C=1 IF ADDRESS CHECK FAILED.
;       C=0 IF ADDRESS CHECK SUCCEEDED.
;
;       R2 =ADDRESS OF WINDOW BLOCK MAPPING BUFFER
;               (FOR PRIV TASKS SEE NOTE.)
;
;       R0 AND R3 ARE PRESERVED ACROSS CALL.
;
;       NOTE :  SINCE PRIVILEGED TASK I /O BUFFERS ARE NOT
;               ADDRESS CHECKED, R2 ALWAYS RETURNS A POINTER TO
;               THE FIRST WINDOW BLOCK.  CHECKPOINTING AND
;               SHUFFLING OF COMMONS WILL STILL WORK PROPERLY
;               PROVIDED THAT A PRIVILEGED TASK NEVER SPECIFIES
;               AN I /O INTO A COMMON WHICH IT ALLOWS TO REMAIN
```

```
        ;               CHECKPOINTABLE AND SHUFFLEABLE.
        ; -
```

Note:

1. In P/OS, almost all drivers will wish to use the alternate routines $CKBFB/$CKBFW which correctly maintain the attachment and partition I/O count mechanism in addition to address checking the user buffer. If the driver completes all references to the buffer in the initiation routine (that is, fills the buffer and calls $IOFIN, rather than queueing the packet and/or starting a transfer which is completed via interrupt service) then it is permissible to use $ACHKB/$ACHCK. See Section 7.5.5 for a description of $CKBFB/$CKBFW and Section 8.1 for an example.

$ALOCB

7.5.2 Allocate Core Buffer

This routine is in the file CORAL.

Calling Sequences:

```
     CALL       $ALOCB
```

or

```
     CALL       $ALOC1
```

Description:

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

$BLKCK
$BLKC1
$BLKC2

7.5.3 Check Logical Block

This routine is in the file MDSUB. The output from this routine is used by disk drivers as input to the $CVLBN routine to handle logical block numbers in data transfers.

Calling Sequence:

```
     CALL      $BLKCK
```

or

```
     CALL      $BLKC2
```

Description:

```
;+
; **-$BLKCK-LOGICAL BLOCK CHECK ROUTINE
; **-$BLKC1-LOGICAL BLOCK CHECK ROUTINE (ALTERNATE ENTRY)
; **-$BLKC2-LOGICAL BLOCK CHECK ROUTINE (ALTERNATE ENTRY FOR
;                QUEUE OPT)
;
; THIS ROUTINE IS CALLED BY I/O DEVICE DRIVERS TO CHECK THE
; STARTING AND ENDING LOGICAL BLOCK NUMBERS OF AN I/O TRANSFER TO
; A FILE STRUCTURED DEVICE. IF THE RANGE OF BLOCKS IS NOT LEGAL,
; THEN $IODON IS ENTERED WITH A FINAL STATUS OF "IE.BLK" AND A
; RETURN TO THE DRIVER'S INITIATOR ENTRY POINT IS EXECUTED. ELSE
; A RETURN TO THE DRIVER IS EXECUTED.
;
; $BLKC2 RETURNS TO $QOPDN IN $DRQRQ IF THERE IS AN ERROR INSTEAD
; OF THE DRIVER'S INITIATOR ENTRY POINT.  THIS ALLOWS THE QUEUE
; OPTIMIZATION CODE TO USE BLKCK
;
; INPUTS:
;
;      R1=ADDRESS OF I/O PACKET.
;      R5=ADDRESS OF THE UCB.
;
; OUTPUTS:
;
;      IF THE CHECK FAILS, THEN $IODON IS ENTERED WITH A FINAL
;      STATUS OF "IE.BLK" AND A RETURN TO THE DRIVER'S INITIATOR
;      ENTRY POINT IS EXECUTED.
;
;      IF THE CHECK SUCCEEDS, THEN THE FOLLOWING REGISTERS ARE
```

```
;       RETURNED:
;               R0=LOW PART OF LOGICAL BLOCK NUMBER.
;               R1=POINTS TO I.PRM+12 (LOW PART OF USER LBN)
;               R2=HIGH PART OF LOGICAL BLOCK NUMBER.
;               R3=ADDRESS OF I/O PACKET.
;-
```

$BLXIO

7.5.4 Move Block of Data

This routine is in file BFCTL.

Calling Sequence:

```
     CALL       $BLXIO
```

Description:

```
     ;+
     ; **-$BLXIO-MOVE BLOCK OF DATA.
     ;
     ; THIS ROUTINE IS CALLED TO MOVE DATA IN MEMORY IN A MAPPED
     ; SYSTEM.
     ;
     ; INPUTS:
     ;
     ;       R0=NUMBER OF BYTES TO MOVE.
     ;       R1=SOURCE APR5 BIAS.
     ;       R2=SOURCE DISPLACEMENT.
     ;       R3=DESTINATION APR6 BIAS.
     ;       R4=DESTINATION DISPLACEMENT.
     ;
     ; OUTPUTS:
     ;
     ;       DESCRIBED MOVE IS ACCOMPLISHED.
     ;       R0 ALTERED
     ;       R1,R3 PRESERVED
     ;       R2,R4 POINT TO LAST BYTE OF SOURCE AND DESTINATION + 1
     ;
     ;       NOTE:   THE COUNT INPUT IN R0 MUST NOT BE ZERO AND IT MUST
     ;               NOT BE LARGE ENOUGH TO CROSS APR BOUNDARIES (THIS
     ;               TYPICALLY MEANS A MAXIMUM OF 8KB-64.BYTES).
     ;-
```

$CKBFI
$CKBFR
$CKBFW
$CKBFB

7.5.5 Check I/O Buffer

These routines are in file EXESB.

Calling Sequences:

```
     CALL      $CKBFB (or appropriate entry name)
```

Description:

```
;+
; **-$CKBFI-CHECK I/O BUFFER FOR I-SPACE (OVERLAY) ACCESS
; **-$CKBFR-CHECK I/O BUFFER FOR READ-ONLY (BYTE) ACCESS
; **-$CKBFW-CHECK I/O BUFFER FOR READ-WRITE (WORD) ACCESS
; **-$CKBFB-CHECK I/O BUFFER FOR READ-WRITE (BYTE) ACCESS
;
; THESE ROUTINES ARE CALLED TO ADDRESS CHECK AN I/O BUFFER
; ASSOCIATED WITH THE CURRENT (UNDER CONSTRUCTION) I/O PACKET.
; IF THE ADDRESS CHECK PASSES, THEN AN ATTEMPT IS MADE TO POINT
; ONE OF THE ATTACHMENT DESCRIPTOR POINTERS AT THE ASSOCIATED
; ADB. THIS WILL HAVE ONE OF THE FOLLOWING OUTCOMES:
;
; 1) - THERE IS CURRENTLY NO ATTACHMENT POINTER IN THE PACKET TO
;      THIS ADB, AND THE POINTERS AREN'T FULL. A POINTER IS FILLED
;      IN AND THE A.IOC, P.IOC FIELDS FOR THIS I/O ARE
;      INCREMENTED. THIS IS THE "NORMAL" SUCCESSFUL CASE.
;
; 2) - THERE IS ALREADY ONE POINTER TO THIS ADB. THE PACKET IS
;      UNTOUCHED, AS ARE THE A.IOC AND P.IOC FIELDS, AND THE CHECK
;      IS CONSIDERED SUCCESSFUL. THE IMPLICATION OF NOT
;      INCREMENTING A.IOC AND P.IOC IS THAT DRIVERS AND ACPS MAY
;      NOT RELEASE BUFFERS FOR AN I/O REQUEST ONE AT A TIME, I.E.
;      THE DRIVER SHOULD NOT CALL $DECIO DIRECTLY, BUT SHOULD CALL
;      $IODON OR $DECAL AFTER ALL BUFFER ACCESS HAS COMPLETED.
;
; 3) - THERE ARE ALREADY TWO POINTERS, NONE OF THEM TO THIS
;      ATTACHMENT DESCRIPTOR. THIS IS CONSIDERED A CHECK FAILURE
;      AND RETURN IS MADE WITH CARRY SET.
;
; INPUTS:
;
;       R0=STARTING ADDRESS OF BLOCK TO BE CHECKED
;       R1=LENGTH OF BUFFER TO BE CHECKED
```

```
;       $ATTPT=ADDRESS OF I.AADA IN CURRENT I/O PACKET
;       HEADER OF THE SUBJECT TASK IS MAPPED THROUGH KISAR6
;
; OUTPUTS:
;
;       C=0 CHECK AND PACKET UPDAT SUCCESSFUL
;               I.AADA OR I.AADA+2 POINTS TO THE ADB
;               A.IOC, P.IOC INCREMENTED
;       C=1 CHECK UNSUCCESSFUL OR PACKET COULD NOT BE FILLED IN
;--
```

$CLINS

7.5.6 Clock Queue Insertion

This routine is in the file QUEUE.

Calling Sequence:

```
CALL      $CLINS
```

Description:

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

7.5.7 Convert Logical Block Number

This routine is in the file MDSUB. The input to this routine is the same as the output from the $BLKCK routine. Typically, a disk driver calls this routine to convert a logical block number to a physical disk address. The routine accesses the U.PRM fields in the driver data base unit control block. These fields contain the sector, track, and cylinder parameters for the type of disk supported. Refer to the description of the U.PRM fields in Section 4.4.4.

Calling Sequence:

```
CALL        $CVLBN
```

Description:

```
;+
; **-$CVLBN-CONVERT LOGICAL BLOCK NUMBER TO DISK PARAMETERS
;
; THIS SUBROUTINE WILL CONVERT THE SPECIFIED LOGICAL BLOCK
; NUMBER TO A SECTOR/TRACK/CYLINDER ADDRESS.
;
; INPUTS:
;
;       (SAME AS $BLKCK OUTPUTS)
;       R0=LOW PART OF LBN
;       R2=HIGH PART OF LBN
;       R3=I/O PACKET ADDRESS
;       R5=UCB ADDRESS
;
; OUTPUTS:
;
;       R0=SECTOR NUMBER
;       R1=TRACK NUMBER
;       R2=CYLINDER NUMBER
;-
```

7.5.8 Deallocate Core Buffer

This routine is in the file CORAL.

Calling sequences:

```
CALL        $DEACB
```

or

```
CALL        $DEAC1
```

Description:

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

7.5.9 Fork

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

Calling sequence:

```
CALL        $FORK
```

Description:

```
;+
; **-$FORK-FORK AND CREATE SYSTEM PROCESS
;
; THIS ROUTINE IS CALLED FROM AN I/O DRIVER TO CREATE A SYSTEM
; PROCESS THAT WILL RETURN TO THE DRIVER AT STACK DEPTH ZERO TO
; FINISH PROCESSING.
;
; INPUTS:
;
;       R5=ADDRESS OF THE UCB FOR THE UNIT BEING PROCESSED.
;       0(SP)=RETURN ADDRESS TO CALLER.
;       2(SP)=RETURN ADDRESS TO CALLERS CALLER.
;
; 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 or $INTSI has run. The fork-processing routine assumes that the Executive has set up entry conditions.

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

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

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

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

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

7.5.10 Fork1

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

Calling Sequence:

```
CALL        $FORK1
```

Description:

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

Notes:

1. A 5-word fork block is required for calls to $FORK1.

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

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

7.5.11 Get Byte

Get Byte is in the file BFCTL. Get Byte manipulates words U.BUF and U.BUF+2 in the UCB.

Calling sequence:

```
CALL        $GTBYT
```

Description:

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

$GTPKT
$GSPKT

7.5.12 Get Packet

Get Packet and Get Special Packet are in the file IOSUB. The recommended way to use $GTPKT is to use the GTPKT$ macro call defined in Section Section 4.3. Usage of $GSPKT is described briefly in Section 1.4.3.

Calling Sequences:

```
     CALL        $GTPKT
```

or

```
     CALL        $GSPKT
```

Description:

```
;+
; **-$GTPKT-GET I/O PACKET FROM REQUEST QUEUE
; **-$GSPKT-GET SELECTIVE I/O PACKET FROM REQUEST QUEUE
;
; THIS ROUTINE IS CALLED BY DEVICE DRIVERS TO DEQUEUE THE NEXT I/O
; REQUEST TO PROCESS. IF THE DEVICE CONTROLLER IS BUSY, THEN A
; CARRY SET INDICATION IS RETURNED TO THE CALLER. ELSE AN ATTEMPT
; IS MADE TO DEQUEUE THE NEXT REQUEST FROM THE CONTROLLER QUEUE.
; IF NO REQUEST CAN BE DEQUEUED, THEN A CARRY SET INDICATION IS
; RETURNED TO THE CALLER. ELSE THE CONTROLLER IS SET BUSY AND A
; CARRY CLEAR INDICATION IS RETURNED TO THE CALLER.
;
; IF QUEUE OPTIMIZATION IS SUPPORTED AND ENABLED FOR THE DEVICE
; THE APROPRIATE PACKET FOR THE CURRENT OPTIMIZATION ALGORITHM
; IS RETURNED.  THREE ALGORITHMS ARE SUPPORTED: NEAREST CYLINDER,
; ELEVATOR, AND C-SCAN.  ALL THREE ALGORITHMS INCORPORATE A
; FAIRNESS COUNT.  IF THE FIRST PACKET ON THE LIST IS PASSED OVER
; MORE THAN "FCOUNT" TIMES, IT IS DONE IMMEDIATELY.
;
;
; THE ALTERNATE ENTRY POINT $GSPKT IS INTENDED FOR USE BY DRIVERS
; WHICH SUPPORT PARALLEL OPERATIONS ON A SINGLE UNIT, A COMMON
; EXAMPLE BEING FULL DUPLEX.  SUCH DRIVERS ARE EXPECTED TO LOOK TO
; THE SYSTEM AS IF THEY ARE ALWAYS FREE, WHILE MAINTAINING THE
; STATUS OF ALL PARALLEL OPERATIONS INTERNALLY WITHIN THEIR OWN
; DEVICE DATA STRUCTURES. PARALLELISM IS ACCOMPLISHED BY HANDLING
; DRIVER-DEFINED CLASSES OF I/O FUNCTION CODES IN PARALLEL WITH
; EACH OTHER.  FOR EXAMPLE A FULL-DUPLEX DRIVER WOULD HANDLE INPUT
```

```
; REQUESTS IN PARALLEL WITH OUTPUT REQUESTS. A DRIVER CALLS $GSPKT
; WHEN IT WANTS TO DEQUEUE A PACKET WHOSE I/O FUNCTION CODE BELONGS
; TO A CERTAIN CLASS.  WHICH FUNCTIONS QUALIFY IS DETERMINED BY AN
; ACCEPTANCE ROUTINE IN THE DRIVER WHOSE ADDRESS IS PASSED TO $GSPKT
; IN R2.  THE ACCEPTANCE ROUTINE IS CALLED BY $GSPKT EACH TIME A
; PACKET IS FOUND IN THE QUEUE WHICH IS ELIGIBLE TO BE DEQUEUED.
; THE ACCEPTANCE ROUTINE IS THEN EXPECTED TO TAKE ONE OF THE
; FOLLOWING THREE ACTIONS:
;
;               1.  RETURN WITH CARRY CLEAR IF THE PACKET SHOULD BE
;                   DEQUEUED.  IN THIS CASE $GSPKT PROCEEDS AS $GTPKT
;                   NORMALLY WOULD ON DEQUEUEING THE PACKET.
;
;               2.  RETURN WITH CARRY SET IF THE PACKET SHOULD NOT BE
;                   DEQUEUED.  IN THIS CASE $GSPKT WILL CONTINUE THE
;                   SCAN OF THE I/O QUEUE.
;
;               3.  ADD THE CONSTANT G$$SPSA TO THE STACK POINTER TO
;                   ABORT THE SCAN WITH NO FURTHER ACTION.
;
; THE ACCEPTANCE ROUTINE MUST SAVE AND RESTORE ANY REGISTERS WHICH
; IT INTENDS TO MODIFY.  WHEN A PACKET IS DEQUEUED VIA $GSPKT, THE
; FOLLOWING NORMAL $GTPKT ACTIONS DO NOT OCCUR:
;
;               1.  FILLING IN OF U.BUF, U.BUF+2 AND U.CNT.  THESE
;                   FIELDS ARE AVAILABLE FOR DRIVER-SPECIFIC USE.
;
;               2.  BUSYING OF UCB AND SCB.
;
;               3.  EXECUTION OF $CFORK TO GET TO PROPER PROCESSOR
;                   (MULTI-PROCESSOR SYSTEMS).
;
; NOTE:             $GSPKT MAY NOT BE USED BY A DRIVER WHICH SUPPORTS
;                   QUEUE OPTIMIZATION.
;
;
; INPUTS:
;
;       R2=ADDRESS OF DRIVER'S ACCEPTANCE ROUTINE (IF CALL AT
;          $GSPKT).
;       R5=ADDRESS OF THE UCB OF THE CONTROLLER TO GET A PACKET
;          FOR.
;
; OUTPUTS:
;
;       C=1 IF CONTROLLER IS BUSY OR NO REQUEST CAN BE DEQUEUED.
;       C=0 IF A REQUEST WAS SUCCESSFULLY DEQUEUED.
;               R1=ADDRESS OF THE I/O PACKET.
;               R2=PHYSICAL UNIT NUMBER.
;               R3=CONTROLLER INDEX.
;               R4=ADDRESS OF THE STATUS CONTROL BLOCK.
```

```
;               R5=ADDRESS OF THE UNIT CONTROL BLOCK.
;
;       NOTE: R4 AND R5 ARE DESTROYED BY THIS ROUTINE.
;-
```

$GTWRD

7.5.13 Get Word

Get Word is in the file BFCTL. It manipulates words U.BUF and U.BUF+2 in the UCB.

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.
     ;-
```

7.5.14 Initiate I/O Buffering

This routine is in the file IOSUB.

Calling Sequence:

```
CALL        $INIBF
```

Description:

```
;+
; **-$INIBF-INITIATE I/O BUFFERING
;
; THIS ROUTINE INITIATES I/O BUFFERING BY DOING THE FOLLOWING:
;
;         1.  DECREMENT THE TASK'S I/O COUNT.
;
;         2.  INCREMENT THE TASK'S BUFFERED I/O COUNT
;
;         3.  INITIATE CHECKPOINTING IF A REQUEST IS PENDING
;
; INPUTS:
;
;         R3=ADDRESS OF I/O PACKET FOR I/O REQUEST.
;
; OUTPUTS:
;
;         R3 IS PRESERVED.
;-
```

$INTXT

7.5.15 Interrupt Exit

Interrupt Exit is in the file SYSXT.

Calling Sequence:

```
     JMP        $INTXT
```

Description:

```
;+
; **-$INTXT-INTERRUPT EXIT
;
; THIS ROUTINE MAY BE CALLED VIA A JMP TO EXIT FROM AN INTERRUPT
;
; INPUTS:
;
;       0(SP)=INTERRUPT SAVE RETURN ADDRESS.
;
; OUTPUTS:
;
;       A RETURN TO INTERRUPT SAVE IS EXECUTED.
;-
```

7.5.16 I/O Done Alternate Entry and I/O Done

These routines are in the file IOSUB.

Calling Sequences:

```
CALL        $IOALT

CALL        $IODON
```

Description:

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

Note:

1. All registers are destroyed when either of these routines is called.

$IOFIN

7.5.17 I/O Finish

I/O Finish is in the file IOSUB. Most drivers do not call I/O Finish, but you should be aware that this routine is executed when a driver calls $IOALT or $IODON. A driver that references an I/O packet before it is queued (bit UC.QUE set--see Section 8.1 for an example) calls I/O Finish if the driver finds an error while preprocessing the I/O packet.

Calling Sequence:

```
     CALL        $IOFIN
```

Description:

```
     ;+
     ; **-$IOFIN-I/O FINISH
     ;
     ; THIS ROUTINE IS CALLED TO FINISH I/O PROCESSING IN CASES WHERE
     ; THE UNIT AND CONTROLLER ARE NOT TO BE DECLARED IDLE. IF THE TASK
     ; WHICH ISSUED THE I/O HAS HAD A RECENT MAPPING CHANGE WHICH MAY
     ; HAVE UNMAPPED ITS I/O STATUS BLOCK, THE I/O PACKET IS QUEUED TO
     ; THE FRONT OF ITS AST QUEUE TO BE COMPLETED LATER IN $FINBF BY
     ; CALLING $IOFIN AGAIN.
     ;
     ; INPUTS:
     ;
     ;       R0=FIRST I/O STATUS WORD.
     ;       R1=SECOND I/O STATUS WORD.
     ;       R3=ADDRESS OF THE I/O REQUEST PACKET.
     ;
     ; 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-ALL ASSOCIATED I/O COUNTS ARE DECREMENTED AND TS.RDN IS
     ;          CLEARED IN CASE THE TASK WAS BLOCKED FOR I/O RUNDOWN.
     ;          T3.MPC IS CLEARED IF THE TASK I/O COUNT GOES TO ZERO TO
     ;          INDICATE THAT THE I/O COUNT WENT TO ZERO AFTER A
     ;          MAPPING CHANGE.
     ;
     ;        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.
;-
```

7.5.18 Put Byte

Put Byte is in the file BFCTL. Put Byte manipulates words U.BUF and U.BUF+2 in the UCB.

Calling Sequence:

```
    CALL        $PTBYT
```

Description:

```
;+
; **-$PTBYT-PUT NEXT BYTE IN USER BUFFER
;
; THIS ROUTINE IS CALLED TO PUT A BYTE IN THE NEXT LOCATION IN THE
; 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.
;-
```

7.5.19 Put Word

Put Word is in the file BFCTL. It manipulates words U.BUF and U.BUF+2 in the UCB.

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
     ; THE 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.
     ;-
```

7.5.20 Queue Insertion by Priority

This routine is in the file QUEUE. A driver may call $QINSP to insert into the I/O queue an I/O packet that the Executive has not already placed in the queue. Queue Insertion by Priority is used only by drivers setting UC.QUE in U.CTL. See Section 8.1 for an example.

Calling Sequence:

```
     CALL        $QINSP
```

Description:

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

7.5.21 Relocate

Relocate is in the file MEMAP. A driver may call $RELOC to relocate a task virtual address while the task is the current task. Relocate is normally used only by drivers setting UC.QUE in U.CTL. See Section 8.1 for an example.

Calling Sequence:

```
     CALL        $RELOC
```

Description:

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

$REQUE
$REQU1

7.5.22 Queue Kernel AST to Task

This routine is in module IOSUB.

Calling Sequence:

```
     CALL        $REQUE
```

or

```
     CALL        $REQU1
```

Description:

```
;+
;**-$REQUE-REQUEUE A REGION LOAD AST TO A TASK AST.
;**-$REQU1-REQUEUE A REGION LOAD AST TO A TASK AST (ALTERNATE
;          ENTRY).
;
; THESE ROUTINES ARE USED TO QUEUE A TASK KERNEL AST WHICH HAS
; BEEN USED AS A REGION LOAD AST BACK AS A TASK AST. THE BUFFERED
; I/O COUNT OF THE TASK IS DECREMENTED IF ENTRY AT $REQUE.
;
; INPUTS:
;      R0=TCB ADDRESS OF ASSOCIATED TASK
;      R3=ADDRESS OF PACKET TO BE QUEUED
;
; OUTPUTS:
;      NONE.
;-
```

7.5.23 Test if Partition Memory Resident for Kernel AST

This routine is in file REQSB.

Calling Sequence:

```
     CALL       $TSPAR
```

Description:

```
;**-$TSPAR-TEST IF PARTITION IS IN MEMORY FOR KERNEL AST
;
; THIS ROUTINE IS CALLED TO CHECK A REGION FOR MEMEORY RESIDENCE
; TO DETERMINE IF IT IS SAFE TO SERVICE A KERNEL AST (E.G. COPY
; A BUFFER) INTO THE REGION. IF THE REGION IS CHECKPOINTED OR
; CURRENTLY BEING CHECKPOINTED, THEN A REGION LOAD AST IS QUEUED
; AND THE REGION IS ACCESSED ON THE TASKS BEHALF.
;
; INPUTS:
;       R0=ADDRESS OF PACKET PEING PROCESSED
;       R1=PCB ADDRESS OF REGION
;       R5=TCB ADDRESS OF ASSOCIATED TASK
;
; OUTPUTS:
;       C=0 IF REGION IS MEMORY RESIDENT
;       C=1 IF REGION IS NON-RESIDENT. IN THIS CASE THE REGION AST
;           HAS BEEN QUEUED, ETC.
;-
```

$TSTBF

7.5.24 Test for I/O Buffering

This routine is in file IOSUB.

Calling Sequence:

```
     CALL       $TSTBF
```

Description:

```
;+
; **-$TSTBF-TEST IF I/O BUFFERING CAN BE INITIATED
;
; THIS ROUTINE DETERMINES IF A GIVEN I/O REQUEST IS ELIGIBLE FOR
; I/O BUFFERING, AND IF SO IT STORES THE PCB ADDRESS OF THE REGION
; INTO WHICH THE TRANSFER IS TO OCCUR IN I.PRM+16 OF THE I/O
; PACKET.
;
; INPUTS:
;
;       R3=ADDRESS OF I/O PACKET FOR I/O REQUEST
;
; OUTPUTS:
;
;       R3 IS PRESERVED.
;
;       C=0 IF I/O BUFFERING CAN BE INITIATED.
;
;       C=1 IF I/O BUFFERING CAN NOT BE INITIATED.
;-
```

7.6 ADDING PHYSICAL MEMORY TO THE P/OS CONFIGURATION

Option modules that contain additional (possibly special purpose) memory, can allow it to be accessible to the system and applications by calling the privileged executive subroutine $STPAR in the distributed PRVLIB.OLB object library. It must be called at system state. In order to access the executive vector table, kernel APR6 is used. Therefore, the routine must be mapped through APR5 when called. This routine is system version independent. Note that system state may be entered without the necessity of being bound to the system version, by using the SWST$ directive.

ADDING PHYSICAL MEMORY TO THE P/OS CONFIGURATION

If the additional memory is general purpose and there are no restrictions on its use, it may be added to the "GEN" main partition. Tasks, commons, and most PLAS regions are allocated from GEN on demand. If the memory is to be used for a dedicated purpose, a main partition of the name "$PARsx" can be specified - where "s" is the logical slot number corresponding to the slot in which the board is located and "x" is any legal RAD50 character. This convention reduces the likelihood of a collision on the partition name. The partition name must be unique among all of the (region and partition) names in the system. Dedicated memory can be accessed, specifying the given main partition name in the creation of a PLAS region subpartition. Memory is allocated from the main partition, using a "first fit" algorithm unless the region is being fixed. If it is, the region is loaded high. If the region is already in memory, an attempt is made to checkpoint it so that it can be loaded high. Once additional memory has been added to the system configuration, it cannot be removed.

If adding memory to the "GEN" partition, do not exit system state until you have turned on the physical memory and configured the base address to the value returned by $STPAR.

The calling interface to $STPAR is as follows:

```
;+
; $STPAR -- add additional memory to system configuration
;
;    Inputs:
;
;    R1 = base address modulus-1 (modulus granularity = 64 bytes)
;         for example, if a base address modulus of 128KB was
;         required, this parameter would be 3777(8).
;    R2 = size of memory to add (granularity = 32 Kbytes)
;    R3 = address of the name of main partition
;           (must be unique if not "GEN   ")
;           (if the partition name "GEN   " is specified, the
;           additional memory will be appended to to this
;           main partition.)
;
```

```
;    Output:
;
;    C=0,
;      R2 = base address of the region (granularity = 64 bytes)
;    C=1,
;      R2 = 0      partition name already exists
;      R2 = 2      insufficient space in available physical
;                  address space
;      R2 = 4      insufficient primary pool space to allocate
;                  required PCBs
;      R2 = 6      $STPAR uses the vectoring scheme described in
;                  this section in addressing several executive
;                  subroutines.  If any of the IDENTS of the
;                  vectored subroutines do not match the IDENTS
;                  that $STPAR expects, $STPAR will fail with
;                  this error code.
;-
This is an example of using the _$STPAR routine.
.b
;+
; This test program will call $STPAR to extend the GEN partition.
;-

     .MCALL DIR$,SWST$,EXIT$S,WIMP$

;
; In order to guarantee that both this code and the code in module
; $STPAR are mapped by APR5 during execution in system state,
; declare this code to reside in the PSECT $STPAQ, which the task
; builder will locate directly preceding PSECT $STPAQ of module
; $STPAR.
;
     .PSECT $STPAQ,I,GBL

START:
;
; Verify that this is a V2.0 or later system.  To do this,
; invoke the WIMP$ directive to retrieve the system version
; number.  Since this form of the WIMP$ directive was not implementec
; before V2.0, if the directive fails, assume that it is not a
; V2.0 system.
;
;
            MOV    #-2,R0    ;ASSUME VERSION # FAILURE
             DIR$  #WIMP     ;GET VERSION
             BCS   5$        ;VERSION NOT AVAILABLE
                             ;MUST BE PRE V2.0
            MOV    #777,R1    ;BIAS MASK (32KB BOUNDARY)
            MOV    #20,R2    ;SIZE OF PARTITION = 2000000 (=512.KB)
                             ;  <16.>*<32KB>
            DIR$   #SWITCH    ;DROP THROUGH THE LOOKING GLASS
```

```
                                   ;(ENTER SYSTEM)
                TST     R0         ;DID AN ERROR OCCUR?
                BEQ     10$        ;IF EQ NO
5$:
                CALL    ERRMSG     ;YES, DO SOMETHING
10$:
                EXIT$S             ;EXAMPLES HAVE NO REAL FUNCTION

SWITCH:                 SWST$   START,STPAR

STPAR:                  CLR     S.WSR0(SP) ;;ASSUME SUCCESS
                CLR     -(SP)     ;;SPECIFY SECOND HALF OF PARTITION NAME
                MOV     #^RGEN,-(SP) ;;SPECIFY FIRST HALF OF PARTITION NAME
                MOV     SP,R3     ;;POINT TO NAME
                CALL    $STPAR    ;;EXPAND GEN
                MOV     (SP)+,(SP)+ ;;POP STACK TWICE WITHOUT AFFECTING
                                  ;;C-BIT
                MOV     R2,S.WSR1(SP) ;;QUALIFY ERROR IF FAILURE
                BCS     10$       ;;IF CS ERROR OCCURRED
;
; The memory has been configured in system data structures.  It must
; be enabled at this point.  Code in module TURNON must also reside in
; same APR5 mapping as this routine and $STPAR.
;
5$:             CALL    TURNON    ;;ENABLE MEMORY AND SET BASE ADDR.
                                  ;;BIAS AT VALUE RETURNED IN R2 BY $STPAR
10$:            BCC     20$
                MOV     #-1,S.WSR0(SP) ;;INDICATE FAILURE TO PROVIDE MEMORY
20$:            RETURN

;
; Turn on memory
;
; Input:        R2 contains base address bias (in units of 64. bytes)
;               of new memory
;
;
TURNON:

; CODE HERE TO ENABLE MEMORY

                RETURN

;
; Error messages
;
; This subroutine runs entirely in user mode, so it should not occupy
; the same PSECT as system state routines.
;
; Input:        R0 = error code
;
```

```
            .PSECT  USRCOD  ;user code psect
ERRMSG:

; ERROR HANDLING

            RETURN

            .PSECT  DBPS,D

WIMP:               WIMP$  GI.FMK,BUF,BUFSIZ    ;GET SYSTEM IDENT

            BUFSIZ = 9.         ;THIS FORM OF WIMP$ RETURNS 9.
                                ;WORDS
BUF:        .BLKW  9.

            .END   START
```

7.1 EXECUTIVE DATA STRUCTURE AND ROUTINE VECTORS

In order to allow greater system version independence for privileged tasks and device drivers, a simple vectoring scheme has been implemented in P/OS version 2.0 and later.

Prior to version 2.0, a privileged task was needed to link with the system's symbol table file to resolve references to various executive routine and data structure addresses. Since these addresses changed with every system version, it was necessary to rebuild the component for each version of the system. By creating a set of absolute addresses which point to various tables, a privileged component can resolve the necessary addresses at runtime.

While it cannot be guaranteed, DIGITAL will attempt to maintain upward compatibility of the P/OS executive data structures and routines. Therefore, the tables provided below are on a "USE AT YOUR OWN RISK" basis. In particular, the data structures may change in future versions of P/OS. Each vectored executive routine is stamped with an IDENT which may be used as a validity check during initialization.

There are three absolute pointers in the P/OS executive. The first points to the executive module LOWCR, the second to SYSCM and the third pointer consists of a bias and a displacement offset by 140000, which is used to map the executive routine vector table. In addition, the $BTMSK bit table has been moved to an absolute location.

7.1.1 Pointer Location and Format

The vector table pointers are located in the following absolute locations:

Symbolic Name	Address	Description
$BTMSK	1002	bitmask table
$VECLC	1042	address of $STACK
$VECSC	1044	address of $CMBEG
$VECVT	1046	bias of vector table
	1050	displacement+140000(8)

The executive routine vector table's format is:

```
                .WORD   number of vectors in table
$VECVT-->       .WORD   $XXX,reserved,IDENT      ;entry point 0
                .WORD   $XXX,reserved,IDENT      ;entry point 1
                                .
                                .
```

where,

$xxx	is the address of the executive routine
reserved	is not currently used though may be used in the future to contain a bias.
IDENT	is a value associated with the particular routine which is incremented when the routine changes in a non upward compatible manner.

7.1.2 Referencing LOWCR and SYSCM Data Structures

The following example illustrates one method of binding an executive data structure reference at runtime.

```
        .MCALL  QIOW$S,EXIT$S   ;system macros

        .PSECT  DATA,D

MYTCB:  .WORD   $TKTCB-$STACK   ;form offset from base of
                                        ;LOWCR to $TKTCB (absolute)
ARGBLK: .BLKW   2               ;argument block/taskname buf
BUF:    .BLKB   80.             ;output buffer

FORMAT: .ASCIZ  /My name is %2R and I got it the hard way/
        .EVEN
```

```
         .PSECT  CODE,I
;
; This task prints its taskname and exits. Could have been
; done simpler using GTSK$ however, the example is one
; technique to resolve the symbol $TKTCB in LOWCR.
;
INIT:    MOV     #ARGBLK,R2        ;point at taskname buffer
         CALL    $SWSTK,TYPIT      ;enter system state
         ADD     @#$VECLC,MYTCB    ;resolve address of $TKTCB
         MOV     MYTCB,R5          ;get current task's TCB addr
         MOV     T.NAM(R5),(R2)+   ;get my task name
         MOV     T.NAM+2(R5),(R2)
         RETURN                    ;return to user state (which
                                          ;restores registers)
TYPIT:   MOV     #BUF,R0           ;point at output buffer
         MOV     #FORMAT,R1        ;point to format string
         MOV     #ARGBLK+2,R2      ;point to argument block
         CALL    $EDMSG            ;format output
         QIOW$S  #IO.WVB,#5,#5,,,,<#BUF,R1,#40> ;output to msg
         EXIT$S                    ;exit

         .END    INIT
```

7.1.3 Referencing Executive Routines

The following example subroutine illustrates one method of referencing a vectored executive routine.

```
         .PSECT  DATA,D

TKTCB:   $TKTCB-$STACK             ;offset in LOWCR to $TKTCB
SETFG:   SETFG$                    ;entry point symbols defined
                                   ; in PRVLIB
         .WORD   1                 ;version number at time routine
                                   ; written
HIVEC:   .WORD   SETFG$/<3*2>      ;each entry point consists of 3
                                   ; words.

         .PSECT  CODE,I
;
; This subroutine sets the this task's local event flag the
; hard way. (It illustrates the vectoring as opposed to a
; SETF$ replacement.)
;
SETF:    CLR     $ERROR
         CALL    $SWSTK,20$        ;enter system state
         ADD     @#VECLC,TKTCB     ;resolve reference to $TKTCB
         MOV     @#KISAR6,-(SP)    ;save current APR6 mapping
```

```
        MOV     @#$VECVT,@#KISAR6 ;map vector table
        MOV     @#VECVT+2,R1     ;point to vector table
        CMP     HIVEC,-2(R1)     ;does vector table describe
                                 ;entry point?
        BHI     10$              ;if hi vectoring error has
                                 ;occurred
        ADD     SETFG,R1         ;point to entry point in table
        CMP     SETFG+2,4(R1)    ;same IDENT?
        BNE     10$              ;if ne no, routine has changed
        MOV     (R1),R1          ;resolve reference to $SETFG
        MOV     #1,R0            ;specify event flag to set
        MOV     @TKTCB,R5        ;get this task's TCB address
        CALL    (R1)             ;set a this task's local event
                                 ;flag 1
        BR      15$
10$:    DEC     $ERROR
15$:    MOV     (SP)+,@#KISAR6
20$:    RETURN
```

7.1.4 Executive Routine Vector Table

```
          .TITLE EXEVEC
          .IDENT /01.00/
 ;
 ; COPYRIGHT (c) 1984 BY DIGITAL EQUIPMENT CORPORATION.
 ; ALL RIGHTS RESERVED.
 ;
 ; THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED
 ; OR COPIED ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE.
 ;
 ;
 ;
 ;+
 ; EXEVEC -- THIS MODULE DEFINES SELECTED EXECUTIVE ENTRY POINTS
 ; SO THAT DRIVERS AND PRIVILEGED TASKS MAY BE SYSTEM
 ; VERSION  INDEPENDENT. THERE IS NO IMPLIED GUARANTEE
 ; THAT THE EACH ENTRY POINT'S INTERFACE WILL REMAIN STABLE
 ; THOUGH THERE IS SOME INTEREST IN KEEPING THEM UPWARD
 ; COMPATIBLE IF POSSIBLE. AN IDENT HAS BEEN PROVIDED
 ; WHICH CAN BE USED TO IDENTIFY THE VERSION OF THE
 ; REFERENCED ROUTINE.
 ;-
   ;
   ; $ALOCB --  ALLOCATE CORE BLOCK
   ;
   ;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = CORAL
   ;   VECTOR TABLE OFFSET NAME = ALOCB$ OFFSET VALUE= 0
   ;
```

```
;
;
; $DEACB --  DEALLOC. CORE BLK
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = CORAL
;   VECTOR TABLE OFFSET NAME = DEACB$ OFFSET VALUE= 6
;
;
;
; $ALOC1 --  ALLOC. CORE BLK (SPECIFIABLE LSTHD)
;
;   ROUTINE VERSION NUMBER   = 1  MODULE NAME = CORAL
;   VECTOR TABLE OFFSET NAME = ALOC1$ OFFSET VALUE= 14
;
;
;
; $DEAC1 --  DEALLOC. CORE BLK (SPECIFIABLE LSTHD)
;
;   ROUTINE VERSION NUMBER   = 1 MODULE NAME = CORAL
;   VECTOR TABLE OFFSET NAME = DEAC1$ OFFSET VALUE= 22
;
;
;
; $ALCLK --  ALLOC. CLOCK QUEUE CORE BLK
;
;   ROUTINE VERSION NUMBER   = 1 MODULE NAME = CORAL
;   VECTOR TABLE OFFSET NAME = ALCLK$ OFFSET VALUE= 30
;
;
;
; $DECLK --  DEALLOC. CLOCK QUEUE CORE BLK
;
;   ROUTINE VERSION NUMBER   = 1 MODULE NAME = CORAL
;   VECTOR TABLE OFFSET NAME = DECLK$ OFFSET VALUE= 36
;
;
;
; $ALPKT --  ALLOC. I/O PACKET
;
;   ROUTINE VERSION NUMBER   = 1 MODULE NAME = CORAL
;   VECTOR TABLE OFFSET NAME = ALPKT$ OFFSET VALUE= 44
;
;
;
; $DEPKT --  DEALLOC. I/O PACKET
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = CORAL
;   VECTOR TABLE OFFSET NAME = DEPKT$ OFFSET VALUE= 52
;
;
;
```

```
;  $ALSEC --  ALLOC. SECONDARY POOL CORE BLK
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = CORAL
;   VECTOR TABLE OFFSET NAME = ALSEC$ OFFSET VALUE= 60
;
;
;
; $DESEC --  DEALLOC. SECONDARY POOL CORE BLK
;
;   ROUTINE   VERSION NUMBER  = 1 MODULE NAME = CORAL
;   VECTOR TABLE OFFSET NAME = DESEC$ OFFSET VALUE= 66
;
;
;
; $GTBYT --  GET NEXT BYTE FROM USER BUFFER
;
;   ROUTINE   VERSION NUMBER  = 1 MODULE NAME = BFCTL
;   VECTOR TABLE OFFSET NAME = GTBYT$ OFFSET VALUE= 74
;
;
;
; $PTBYT --  PUT NEXT BYTE IN USER BUFFER
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = BFCTL
;   VECTOR TABLE OFFSET NAME = PTBYT$ OFFSET VALUE= 102
;
;
;
; $GTWRD --  GET NEXT WORD FROM USER BUFFER
;
;   ROUTINE   VERSION NUMBER   = 1  MODULE NAME = BFCTL
;   VECTOR TABLE OFFSET NAME = GTWRD$  OFFSET VALUE= 110
;
;
;
; $PTWRD --  PUT NEXT WORD IN USER BUFFER
;
;   ROUTINE   VERSION NUMBER  = 1 MODULE NAME = BFCTL
;   VECTOR TABLE OFFSET NAME = PTWRD$ OFFSET VALUE= 116
;
;
;
; $BLXIO --  MOVE BLOCK OF DATA
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = BFCTL
;   VECTOR TABLE OFFSET NAME = BLXIO$  OFFSET VALUE= 124
;
;
;
; $CLINS --  CLOCK QUEUE INSERTION
;
```

```
;   ROUTINE   VERSION NUMBER    = 1 MODULE NAME = QUEUE
;   VECTOR TABLE OFFSET NAME = CLINS$  OFFSET VALUE= 132
;
;
;
; $CLRMV --  CLOCK QUEUE REMOVAL
;
;   ROUTINE   VERSION NUMBER    = 1  MODULE NAME = QUEUE
;   VECTOR TABLE OFFSET NAME = CLRMV$  OFFSET VALUE= 140
;
;
;
; $QINSF --  QUEUE INSERTION AT END OF LIST
;
;   ROUTINE   VERSION NUMBER    = 1 MODULE NAME = QUEUE
;   VECTOR TABLE OFFSET NAME = QINSF$  OFFSET VALUE= 146
;
;
;
; $QINSP --  QUEUE INSERTION BY PRIORITY
;
;   ROUTINE   VERSION NUMBER    = 1  MODULE NAME = QUEUE
;   VECTOR TABLE OFFSET NAME = QINSP$  OFFSET VALUE= 154
;
;
;
; $QINSB --  QUEUE INSERTION AT BEGINNING
;
;   ROUTINE   VERSION NUMBER    = 1  MODULE NAME = QUEUE
;   VECTOR TABLE OFFSET NAME = QINSB$  OFFSET VALUE= 162
;
;
;
; $QRMVA --  QUEUE REMOVAL BY BLOCK ADDRESS
;
;   ROUTINE   VERSION NUMBER    = 1  MODULE NAME = QUEUE
;   VECTOR TABLE OFFSET NAME = QRMVA$  OFFSET VALUE= 170
;
;
;
; $QRMVT --  QUEUE REMOVAL BY TCB ADDRESS
;
;   ROUTINE   VERSION NUMBER    = 1  MODULE NAME = QUEUE
;   VECTOR TABLE OFFSET NAME = QRMVT$  OFFSET VALUE= 176
;
;
;
; $QSPIB --  QUEUE INSERTION (SEC. POOL) AT BEG.
;
;   ROUTINE   VERSION NUMBER    = 1  MODULE NAME = QUEUE
;   VECTOR TABLE OFFSET NAME = QSPIB$  OFFSET VALUE= 204
```

```
;
;
;
; $QSPIF --  QUEUE INSERTION (SEC. POOL) AT END.
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = QUEUE
;   VECTOR TABLE OFFSET NAME = QSPIF$  OFFSET VALUE= 212
;
;
;
; $QSPRF --  QUEUE REMOVAL (SEC. POOL) FROM FRONT
;
;   ROUTINE   VERSION NUMBER   = 1  MODULE NAME = QUEUE
;   VECTOR TABLE OFFSET NAME = QSPRF$  OFFSET VALUE= 220
;
;
;
; $QSPIP --  QUEUE INSERTION (SEC. POOL) BY PRI.
;
;   ROUTINE   VERSION NUMBER   = 1  MODULE NAME = QUEUE
;   VECTOR TABLE OFFSET NAME = QSPIP$  OFFSET VALUE= 226
;
;
;
; $GTSPK --  QUEUE REMOVAL (SEC. POOL) BY BLK ADR
;
;   ROUTINE   VERSION NUMBER   = 1  MODULE NAME = QUEUE
;   VECTOR TABLE OFFSET NAME = GTSPK$  OFFSET VALUE= 234
;
;
;
; $ACHCK --  ADDRESS CHECK WORD ALIGNED (NO I/O CNTS)
;
;   ROUTINE   VERSION NUMBER   = 1  MODULE NAME = EXESB
;   VECTOR TABLE OFFSET NAME = ACHCK$  OFFSET VALUE= 242
;
;
;
; $ACHKB --  ADDRESS CHECK BYTE ALIGNED (NO I/O CNTS)
;
;   ROUTINE   VERSION NUMBER   = 1  MODULE NAME = EXESB
;   VECTOR TABLE OFFSET NAME = ACHKB$  OFFSET VALUE= 250
;
;
;
; $ACHRO --  ADDRESS CHECK FOR READONLY ACCESS NO IOC
;
;   ROUTINE   VERSION NUMBER   = 1  MODULE NAME = EXESB
;   VECTOR TABLE OFFSET NAME = ACHRO$  OFFSET VALUE= 256
;
;
```

```
;
; $CKBFW --  CHECK I/O BUFFER FOR READONLY (BYTE)
;
;   ROUTINE   VERSION NUMBER    = 1 MODULE NAME = EXESB
;   VECTOR TABLE OFFSET NAME = CKBFW$ OFFSET VALUE= 264
;
;
;
; $CKBFB --  CHECK I/O BUFFER FOR READWRITE (BYTE)
;
;   ROUTINE   VERSION NUMBER    = 1 MODULE NAME = EXESB
;   VECTOR TABLE OFFSET NAME = CKBFB$ OFFSET VALUE= 272
;
;
;
; $CKBFR --  CHECK I/O BUFFER FOR READWRITE (WORD)
;
;   ROUTINE   VERSION NUMBER    = 1 MODULE NAME = EXESB
;   VECTOR TABLE OFFSET NAME = CKBFR$ OFFSET VALUE= 300
;
;
;
; $CEFIG --  CONVERT EVENT FLAG AND LOCK FOR I/O
;
;   ROUTINE   VERSION NUMBER    = 1 MODULE NAME = EXESB
;   VECTOR TABLE OFFSET NAME = CEFIG$ OFFSET VALUE= 306
;
;
;
; $CEFI --  CONVERT EVENT FLAG FOR I/O
;
;   ROUTINE   VERSION NUMBER    = 1 MODULE NAME = EXESB
;   VECTOR TABLE OFFSET NAME = CEFI$ OFFSET VALUE= 314
;
;
;
; $CVDVN --  CONVERT DEV NAM AND LOGICAL UNIT TO UCB
;
;   ROUTINE   VERSION NUMBER    = 1 MODULE NAME = EXESB
;   VECTOR TABLE OFFSET NAME = CVDVN$ OFFSET VALUE= 322
;
;
;
; $MPLNE --  MAP LOGICAL UNIT NUMBER
;
;   ROUTINE   VERSION NUMBER    = 1 MODULE NAME = EXESB
;   VECTOR TABLE OFFSET NAME = MPLNE$ OFFSET VALUE= 330
;
;
;
; $MPLND --  MAP LOGICAL UNIT NUMBER (ALTRN. ENTRYPT)
```

```
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = EXESB
;   VECTOR TABLE OFFSET NAME = MPLND$ OFFSET VALUE= 336
;
;
;
; $TKWSE --  WAIT FOR SIGNIFICANT EVENT
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = EXESB
;   VECTOR TABLE OFFSET NAME = TKWSE$ OFFSET VALUE= 344
;
;
;
; $RELOC --  RELOCATE USER VIRTUAL ADDRESS
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = MEMAP
;   VECTOR TABLE OFFSET NAME = RELOC$ OFFSET VALUE= 352
;
;
;
; $RELOM --  RELOCATE AND MAP USER VIRTUAL ADDRESS
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = MEMAP
;   VECTOR TABLE OFFSET NAME = RELOM$ OFFSET VALUE= 360
;
;
;
; $CVLBN --  CONVERT LOGICAL BLOCK NUMBER TO DISK PARAMS
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = MDSUB
;   VECTOR TABLE OFFSET NAME = CVLBN$ OFFSET VALUE= 366
;
;
;
; $BLKCK --  LOGICAL BLOCK CHECK ROUTINE
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = MDSUB
;   VECTOR TABLE OFFSET NAME = BLKCK$ OFFSET VALUE= 374
;
;
;
; $BLKC1 --  LOGICAL BLOCK CHECK ROUTINE ALTRN. ENTRYPT
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = MDSUB
;   VECTOR TABLE OFFSET NAME = BLKC1$ OFFSET VALUE= 402
;
;
;
; $BLKC2 --  LOGICAL BLOCK CHECK ROUTINE FOR QUEUE OPT
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = MDSUB
```

```
;   VECTOR TABLE OFFSET NAME = BLKC2$ OFFSET VALUE= 410
;
;
;
; $RQCNC --  REQUEST CONTROLLER FOR CONTROL OPERATION
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = MDSUB
;   VECTOR TABLE OFFSET NAME = RQCNC$ OFFSET VALUE= 416
;
;
;
; $RQCND --  REQUEST CONTROLLER FOR DATA TRANSFER
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = MDSUB
;   VECTOR TABLE OFFSET NAME = RQCND$ OFFSET VALUE= 424
;
;
;
; $RLCN --  RELEASE CONTROLLER
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = MDSUB
;   VECTOR TABLE OFFSET NAME = RLCN$ OFFSET VALUE= 432
;
;
;
; $VOLVD --  PREPROCESS VOLUME VALID FUNCTION
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = MDSUB
;   VECTOR TABLE OFFSET NAME = VOLVD$ OFFSET VALUE= 440
;
;
;
; $VOLSC --  VOLUME STATUS CHANGE
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = OLRSR
;   VECTOR TABLE OFFSET NAME = VOLSC$ OFFSET VALUE= 446
;
;
;
; $DECIO --  DECREMENT I/O COUNT VIA ADB
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = IOSUB
;   VECTOR TABLE OFFSET NAME = DECIO$ OFFSET VALUE= 454
;
;
;
; $DECIP --  DECREMENT I/O COUNT PARTITION ONLY
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = IOSUB
;   VECTOR TABLE OFFSET NAME = DECIP$ OFFSET VALUE= 462
;
```

```
;
;
; $GTPKT --  GET I/O PACKET FROM REQUEST QUEUE
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = IOSUB
;   VECTOR TABLE OFFSET NAME = GTPKT$ OFFSET VALUE= 470
;
;
;
; $GSPKT --  GET SELECTIVE I/O PACKET FROM REQUEST QUEUE
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = IOSUB
;   VECTOR TABLE OFFSET NAME = GSPKT$ OFFSET VALUE= 476
;
;
;
; $TSTBF --  TEST IF I/O BUFFERING SHOULD BE INITIATED
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = IOSUB
;   VECTOR TABLE OFFSET NAME = TSTBF$ OFFSET VALUE= 504
;
;
;
; $INIBF --  INITIATE I/O BUFFERING
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = IOSUB
;   VECTOR TABLE OFFSET NAME = INIBF$ OFFSET VALUE= 512
;
;
;
; $QUEBF --  QUEUE BUFFERED I/O FOR COMPLETION
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = IOSUB
;   VECTOR TABLE OFFSET NAME = QUEBF$ OFFSET VALUE= 520
;
;
;
; $REQUE --  REQUEUE A REGION LOAD AST TO A TASK AST
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = IOSUB
;   VECTOR TABLE OFFSET NAME = REQUE$ OFFSET VALUE= 526
;
;
;
; $REQU1 --  REQUEUE A REG LOAD AST (ALTERNATE ENTRYPOINT)
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = IOSUB
;   VECTOR TABLE OFFSET NAME = REQU1$ OFFSET VALUE= 534
;
;
;
```

```
; $IOALT --  I/O DONE ALTERNATE ENTRY POINT (ERRORS)
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = IOSUB
;   VECTOR TABLE OFFSET NAME = IOALT$ OFFSET VALUE= 542
;
;
;
; $IODON --  I/O DONE
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = IOSUB
;   VECTOR TABLE OFFSET NAME = IODON$ OFFSET VALUE= 550
;
;
;
; $IOFIN --  I/O FINISH
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = IOSUB
;   VECTOR TABLE OFFSET NAME = IOFIN$ OFFSET VALUE= 556
;
;
;
; $DECAL --  DECREMENT ALL I/O COUNTS AND UNBLK TASK.
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = IOSUB
;   VECTOR TABLE OFFSET NAME = DECAL$ OFFSET VALUE= 564
;
;
;
; $DECBF --  DECREMENT ALL PARTITION I/O CNTS & UNBLK TASK
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = IOSUB
;   VECTOR TABLE OFFSET NAME = DECBF$ OFFSET VALUE= 572
;
;
;
; $IOKIL --  I/O KIL
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = IOSUB
;   VECTOR TABLE OFFSET NAME = IOKIL$ OFFSET VALUE= 600
;
;
;
; $SCDVT --  SCAN DEVICE TABLES
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = IOSUB
;   VECTOR TABLE OFFSET NAME = SCDVT$ OFFSET VALUE= 606
;
;
;
; $SCDV1 --  SCAN DEVICE TABLES (ALTRN. ENTRYPT)
;
```

```
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = IOSUB
;   VECTOR TABLE OFFSET NAME = SCDV1$ OFFSET VALUE= 614
;
;
;
; $SRNAM --  SEARCH FOR NAMED PARTITION
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = PLSUB
;   VECTOR TABLE OFFSET NAME = SRNAM$ OFFSET VALUE= 622
;
;
;
; $SETCR --  SET CONDITIONAL SCHEDULE REQUEST
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = SETCR$ OFFSET VALUE= 630
;
;
;
; $SETRQ --  SET SCHEDULE REQUEST
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = SETRQ$ OFFSET VALUE= 636
;
;
;
; $SETRT --  SET SCHEDULE REQUEST FOR CURRENT TASK
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = SETRT$ OFFSET VALUE= 644
;
;
;
; $SETMG --  SET EVENT FLG & UNLCK W/EVNT FLG MASK
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = SETMG$ OFFSET VALUE= 652
;
;
;
; $SETFG --  SET EVENT FLAG AND UNLOCK W/EFN NUMBER
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = SETFG$ OFFSET VALUE= 660
;
;
;
; $DASTT --  DECLARE AST TRAP
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = DASTT$ OFFSET VALUE= 666
```

```
;
;
;
; $QASTC --  QUEUE AST TO TASK (USED  W/CINT$)
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = QASTC$ OFFSET VALUE= 674
;
;
;
; $QASTT --  QUEUE AST TO TASK
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = QASTT$ OFFSET VALUE= 702
;
;
;
; $SRSTD --  SEARCH SYSTEM TASK DIRECTORY
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = SRSTD$ OFFSET VALUE= 710
;
;
;
; $STPCT --  STOP CURRENT TASK
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = STPCT$ OFFSET VALUE= 716
;
;
;
; $STPTK --  STOP TASK
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = STPTK$ OFFSET VALUE= 724
;
;
;
; $NXTSK --  ASSIGN NEXT REGION TO PARTITION
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = NXTSK$ OFFSET VALUE= 732
;
;
;
; $TSPAR --  TEST IF PARTITION IN MEMORY FOR KERNEL AST
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = TSPAR$ OFFSET VALUE= 740
;
;
```

```
;
; $ICHKP --  INITIATE CHECKPOINT
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = ICHKP$ OFFSET VALUE= 746
;
;
;
; $EXRQP --  EXEC REQUEST WITH QUEUE INSERT BY PRIORITY
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = EXRQP$ OFFSET VALUE= 754
;
;
;
; $EXRQF --  EXEC REQUEST WITH QUEUE INSERT FIFO
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = EXRQF$ OFFSET VALUE= 762
;
;
;
; $EXRQN --  EXEC REQUEST WITH NO QUEUE INSERT
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = EXRQN$ OFFSET VALUE= 770
;
;
;
; $EXRQU --  EXEC REQUEST AND UNSTOP WITH NO INSERT
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = EXRQU$ OFFSET VALUE= 776
;
;
;
; $EXRQS --  EXEC REQUEST WITH NO SCHEDULE REQUEST
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = EXRQS$ OFFSET VALUE= 1004
;
;
;
; $TSKRT --  TASK REQUEST (DEFAULT UCB)
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = TSKRT$ OFFSET VALUE= 1012
;
;
;
; $TSKRQ --  TASK REQUEST (SPECIFY UCB)
```

```
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = TSKRQ$ OFFSET VALUE= 1020
;
;
;
; $TSKRP --  TASK REQUEST (SPECIFY DEFAULT UIC)
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = REQSB
;   VECTOR TABLE OFFSET NAME = TSKRP$ OFFSET VALUE= 1026
;
;
;
; $FORK --  FORK AND CREATE SYSTEM PROCESS
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = SYSXT
;   VECTOR TABLE OFFSET NAME = FORK$ OFFSET VALUE= 1034
;
;
;
; $FORK1 --  FORK AND CREATE SYSTEM PROCESS
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = SYSXT
;   VECTOR TABLE OFFSET NAME = FORK1$ OFFSET VALUE= 1042
;
;
;
; $FORK0 --  FORK AND CREATE SYSTEM PROCESS
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = SYSXT
;   VECTOR TABLE OFFSET NAME = FORK0$ OFFSET VALUE= 1050
;
;
;
; $FORK2 --  FORK AND CREATE SYSTEM PROCESS USED W/CINT$
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = SYSXT
;   VECTOR TABLE OFFSET NAME = FORK2$ OFFSET VALUE= 1056
;
;
;
; $QFORK --  INSERT FORK BLOCK AT END OF FORK QUEUE
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = SYSXT
;   VECTOR TABLE OFFSET NAME = QFORK$ OFFSET VALUE= 1064
;
;
;
; $NONSI --  NONSENSE INTERRUPT ENTRY POINT
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = SYSXT
```

```
;   VECTOR TABLE OFFSET NAME = NONSI$ OFFSET VALUE= 1072
;
;
;
; $DSPKA --  DISPATCH KERNEL AST
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = SYSXT
;   VECTOR TABLE OFFSET NAME = DSPKA$ OFFSET VALUE= 1100
;
;
;
; $SGFIN --  SEGMENT FAULT AND TRAP 4 INTERCEPT ROUTINE
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = SYSXT
;   VECTOR TABLE OFFSET NAME = SGFIN$ OFFSET VALUE= 1106
;
;
;
; $NLTMO --  NULL TIMEOUT ROUTINE
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = TDSCH
;   VECTOR TABLE OFFSET NAME = NLTMO$ OFFSET VALUE= 1114
;
;
;
; $MUL --  INTEGER MULTIPLY MAGNITUDE NUMBERS
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = UTSUB
;   VECTOR TABLE OFFSET NAME = MUL$ OFFSET VALUE= 1122
;
;
;
; $DIV --  INTEGER DIVIDE MAGNITUDE NUMBERS
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = UTSUB
;   VECTOR TABLE OFFSET NAME = DIV$ OFFSET VALUE= 1130
;
;
;
; $DBDIV --  DOUBLE PRECISION DIVIDE MAGNITUDE NUMBERS
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = UTSUB
;   VECTOR TABLE OFFSET NAME = DBDIV$ OFFSET VALUE= 1136
;
;
;
; $CAT5 --  CONVERT ASCII TO RAD50
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = UTSUB
;   VECTOR TABLE OFFSET NAME = CAT5$ OFFSET VALUE= 1144
;
```

```
;
;
; $SAVNR --  SAVE NON VOLATILE REGISTERS
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = UTSUB
;   VECTOR TABLE OFFSET NAME = SAVNR$ OFFSET VALUE= 1152
;
;
;
; $DRQRQ --  QUEUE I/O REQUEST (INTERNAL ENTRYPT)
;
;   ROUTINE   VERSION NUMBER   = 1 MODULE NAME = DRSUB
;   VECTOR TABLE OFFSET NAME = DRQRQ$ OFFSET VALUE= 1160
;
;
```

CHAPTER 8

HANDLING SPECIAL USER BUFFERS

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

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

8.1 DRIVER CODE

The coding shown in this chapter is an excerpt from a driver that illustrates the handling of a special user buffer. The key points are:

1. The UC.QUE bit has been set in the control byte (U.CTL) of the UCB for each device/unit.
2. The routine (ZZINI) that is defined as the I/O initiation entry point in the driver dispatch table (DDT$) macro call performs the following actions:
    1. Retrieves the user virtual address and address-checks it
    2. Relocates the virtual address and stores the result back into the packet
    3. Inserts the packet into the I/O queue and continues execution inline to the entry point BMINI, which calls $GTPKT

3. The driver propagates its own execution by branching back to BMINI to call $GTPKT.

```
        .TITLE  BMTAB - DATA BASE FOR BLOCK MOVE DRIVER
        .IDENT  /01/
;
;           COPYRIGHT (c) 1981, 1982 BY
;       DIGITAL EQUIPMENT CORPORATION, MAYNARD
;        MASSACHUSETTS.  ALL RIGHTS RESERVED.
;
;THIS  SOFTWARE  IS  FURNISHED  UNDER  A LICENSE AND MAY BE USED
;AND  COPIED  ONLY IN  ACCORDANCE WITH THE TERMS OF SUCH LICENSE
;AND WITH  THE INCLUSION  OF THE ABOVE  COPYRIGHT  NOTICE.  THIS
;SOFTWARE  OR ANY OTHER  COPIES  THEREOF, MAY NOT BE PROVIDED OR
;OTHERWISE MADE  AVAILABLE TO ANY OTHER PERSON.  NO TITLE TO AND
;OWNERSHIP OF THE SOFTWARE IS HEREBY TRANSFERRED.
;
;THE INFORMATION  IN THIS DOCUMENT IS SUBJECT  TO CHANGE WITHOUT
;NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT  BY  DIGITAL
;EQUIPMENT CORPORATION.
;
;DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
;ITS SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY DIGITAL.
;
;
;
;
;
;LOADABLE DATA BASE FOR EXAMPLE BUFFERED I/O DRIVER
;
;MACRO LIBRARY CALLS
;
        .MCALL  CLKDF$
        .MCALL  HWDDF$
        .MCALL  SCBDF$
        .MCALL  UCBDF$

        CLKDF$                  ;DEFINE CLOCK BLOCK OFFSETS
        HWDDF$                  ;DEFINE HARDWARE REGISTERS
        SCBDF$  ,,SYSDEF        ;DEFINE SCB OFFSETS
        UCBDF$                  ;DEFINE UCB OFFSETS

$BMDAT::
;
;                                             BM DCB
;
```

DRIVER CODE

```
$BMDCB::

          .WORD    0                   ; D.LNK
          .WORD    .BM0                ; D.UCB
          .ASCII   /BM/                ; D.NAM
          .BYTE    0,1-1               ; D.UNIT,D.UNIT+1
          .WORD    BMND-BMST           ; D.UCBL
          .WORD    $0                  ; D.DSP
                                       ; D.MSK - FUNCTION MASKS
          .WORD    33                  ; LEGAL    0-17 IO.KIL,IO.WLB,IO.ATT
                                       ;               IO.DET
          .WORD    31                  ; CONTROL  0-17 IO.KIL,IO.ATT,IO.DET
          .WORD    0                   ; NOOP     0-17
          .WORD    0                   ; ACP      0-17
          .WORD    4                   ; LEGAL   20-37 IO.WVB
          .WORD    0                   ; CONTROL 20-37
          .WORD    0                   ; NOOP    20-37
          .WORD    0                   ; ACP     20-37
          .WORD    0                   ; D.PCB

;                                                    BM UCB'S
;
;

          PR0=0


;

          .IF      DF M$$MUP
          .WORD    0
          .ENDC


.BM0::
          .WORD    $BMDCB              ; U.DCB
          .WORD    .-2                 ; U.RED
          .BYTE    UC.QUE,0            ; U.CTL,U.STS
          .BYTE    0,US.OFL            ; U.UNIT,U.ST2
          .WORD    DV.REC              ; U.CW1
          .WORD    0                   ; U.CW2
          .WORD    0                   ; U.CW3
          .WORD    72.                 ; U.CW4
          .WORD    $BM0                ; U.SCB
          .WORD    0                   ; U.ATT
          .WORD    0,0                 ; U.BUF,U.BUF+2
          .WORD    0                   ; U.CNT

BMND=.
```

DRIVER CODE

```
;
;                                                       BM SCB'S
;
;

$BM0::  .WORD   0,.-2               ; S.LHD
        .WORD   0,0,0,0             ; S.FRK
        .WORD   0                   ; S.KS5
        .WORD   0                   ; S.PKT
        .BYTE   0,0                 ; S.CTM,S.ITM
        .BYTE   0,0                 ; S.STS,S.ST3
        .WORD   0                   ; S.ST2
        .WORD   0                   ; S.KRB - NO KRB SINCE NO CONTROLLER

$BMEND::

        .END
        .TITLE  BMDRV - BLOCK MOVE DRIVER
        .IDENT  /01/

;+
;COPYRIGHT (c) 1981,1982 BY DIGITAL EQUIPMENT CORPORATION.
;ALL RIGHTS RESERVED.
;
;THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED
;OR COPIED ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE.
;
;
;
;
;
;       THIS IS A SAMPLE DRIVER WHICH DEMONSTRATES HOW TO USE SOME
;       OF THE MORE SOPHISTICATED EXECUTIVE SERVICES AVAILABLE TO
;       I/O DRIVERS. THIS DRIVER DEMONSTRATES:
;
;       1) THE CHECKING OF ADDITIONAL USER BUFFERS PRIOR TO QUEUEING
;          AN I/O PACKET.
;
;       2) USE OF THE CLOCK QUEUE FROM A DRIVER.
;
;       3) USE OF THE BUFFERED I/O MECHANISM
;
;       4) USE OF THE GENERAL BUFFERED I/O KERNEL AST MECHANISM
;
;       5) USE OF REGION LOAD KERNEL ASTS
;
;       6) USE OF BLXIO
;
;
;       THIS DRIVER UNDERSTANDS PRECISELY ONE QIO, WHICH IS:
;
```

DRIVER CODE

```
;       ...     IO.WLB,.....,<DEST-BUFFER,LENGTH,TIME,SRC-BUFFER>
;                 OR
;               IO.WVB
;
;       THE DRIVER QUEUES A CLOCK BLOCK FOR TIME TICKS AND AT THE
;       END OF THAT TIME INTERVAL COPIES THE SOURCE BUFFER TO THE
;       DESTINATION BUFFER. IF POSSIBLE, THE REQUEST IS BUFFERED
;       INTERNALLY WHILE THE CLOCK REQUEST IS POSTED.
;-


        .MCALL  CLKDF$, PKTDF$

        CLKDF$                  ;DEFINE CLOCK BLOCK OFFSETS
        PKTDF$                  ;DEFINE I/O PACKET OFFSETS


;
;       DEFINE MAXIMUM TRANSFER LENGTH WHICH WILL BE BUFFERED
;


        BUFLIM = 100.


        DDT$    BM,,NONE,,,NEW


;+
;** - BMINI - I/O INITIATION ENTRY POINT
;
;
;INPUTS:
;
;       DRQIO (BECAUSE THE UC.QUE BIT IS SET IN THE UCB) SETS THE
;       REGISTERS TO THE FOLLOWING:
;
;       R1 = ADDRESS OF I/O PACKET
;       R4 = ADDRESS OF SCB
;       R5 = ADDRESS OF UCB
;
;OUTPUTS:
;
;       IF THE SPECIFIED CONTROLLER IS NOT BUSY AND AN I/O REQUEST IS
;       WAITING TO BE PROCESSED, THEN THE REQUEST IS DEQUEUED AND THE
;       I/O OPERATION IS INITIATED.
;
;       I/O REQUEST PACKET FORMAT:
;
;       I.LNK       -- I/O QUEUE THREAD WORD.
;       I.PRI/I.EFN -- REQUEST PRIORITY, EVENT FLAG NUMBER.
```

```
;       I.TCB         -- ADDRESS OF THE TCB OF THE REQUESTER TASK.
;       I.LN2         -- POINTER TO SECOND LUN WORD IN REQUESTER TASK
;                        HEADER.
;       I.UCB         -- UCB ADDRESS OF DEVICE
;       I.FCN         -- I/O FUNCTION CODE (IO.WLB).
;       I.IOSB        -- VIRTUAL ADDRESS OF I/O STATUS BLOCK.
;       I.IOSB+2      -- RELOCATION BIAS OF I/O STATUS BLOCK.
;       I.IOSB+4      -- I/O STATUS BLOCK ADDRESS (DISPLACEMENT +
;                        140000).
;       I.IOSB+6      -- VIRTUAL ADDRESS OF AST SERVICE ROUTINE.
;       I.PRM         -- RELOCATION BIAS OF SOURCE BUFFER.
;       I.PRM+2       -- BUFFER ADDRESS OF I/O TRANSFER.
;       I.PRM+4       -- NUMBER OF BYTES TO BE TRANSFERED.
;       I.PRM+6       -- TIME DISPLACEMENT IN TICKS
;       I.PRM+10      -- VIRTUAL ADDRESS (TO BECOME RELOCATION BIAS) OF
;                        DESTINATION BUFFER
;       I.PRM+12      -- FILLED IN WITH DISPLACEMENT ADDRESS OF
;                        DESTINATION BUFFER
;       I.PRM+14      -- USED TO STORE BUFFER/CLOCK BLOCK ADDRESS
;       I.PRM+16      -- FILLED IN WITH PCB ADDRESS OF OUTPUT BUFFER
;-


         .ENABL  LSB


;       *************************************************************
;       *                                                           *
;       *       I N I T I A T I O N   E N T R Y   P O I N T         *
;       *                                                           *
;       *************************************************************

BMINI:                              ; PRE-QUEUING INITIALIZE ENTRY POINT


;       *************************************************************
;       *                                                           *
;       *     ADDRESS CHECK THE SOURCE BUFFER WHILE THE TASKS       *
;       *     CONTEXT IS LOADED, AND FILL IN THE NECESSARY          *
;       *     PARAMETERS IN THE I/O PACKET                          *
;       *                                                           *
;       *************************************************************

         MOV     R1,R3              ; COPY ADDRESS OF I/O PACKET
         MOV     I.PRM+10(R1),R0    ; GET VIRTUAL ADDRESS OF SOURCE
                                    ; BUFFER
         MOV     I.PRM+4(R3),R1     ; AND LENGTH OF SOURCE BUFFER

;       +-----------------------------------------------------------+
;       |                                                           |
;       |       THE INPUT PARAMETERS FOR $CKBFR ARE:                |
```

```
;      |                                                            |
;      |       R0 = STARTING ADDRESS OF BLOCK TO BE CHECKED         |
;      |       R1 = LENGTH OF THE BLOCK TO BE CHECKED               |
;      |       $ATTPT = ADDRESS OF I.AADA IN I/O PACKET             |
;      |                (ESTABLISHED IN DRQIO)                      |
;      |       CURRENT TASK HEADER MUST BE MAPPED THROUGH APR 6     |
;      |                (ESTABLISHED BY DIRECTIVE DISPATCHER)       |
;      |                                                            |
;      |       THE OUTPUT PARAMETERS ARE:                           |
;      |                                                            |
;      |       C = 0 IF CHECK AND PACKET UPDATE SUCCESSFUL          |
;      |             I.AADA OR I.AADA IN PACKET POINTS TO           |
;      |             RELATED ADB, P.IOC, A.IOC INCREMENTED          |
;      |       C = 1 IF CHECK UNSUCCESFUL OR I.AADA, I.AADA         |
;      |             ALREADY FILLED IN                              |
;      |                                                            |
;      +------------------------------------------------------------+

        CALL    $CKBFR          ; CHECK BUFFER, INCREMENT A.IOC AND
                                ; P.IOC FOR APPROPRIATE REGIONS
        BCC     10$             ; IF CC ALL WAS OK

;      *************************************************************
;      *                                                           *
;      *       SOURCE BUFFER WAS ILLEGAL, FINISH I/O HERE          *
;      *                                                           *
;      *************************************************************

        MOV     #IE.SPC&377,R0  ; SET COMPLETION STATUS
        CLR     R1              ; AND NUMBER OF BYTES TRANSFERRED

;
;      +-----------------------------------------------------------+
;      |                                                           |
;      |       THE INPUT PARAMETERS FOR $IOFIN ARE:                |
;      |                                                           |
;      |       R0 = FIRST  WORD OF I/O STATUS TO RETURN            |
;      |       R1 = SECOND WORD OF I/O STATUS TO RETURN            |
;      |       R3 = ADDRESS OF I/O PACKET                          |
;      |                                                           |
;      |       THE OUTPUT PARAMETERS ARE:                          |
;      |                                                           |
;      |       R4 IS DESTROYED                                     |
;      |                                                           |
;      +-----------------------------------------------------------+
;

        CALLR   $IOFIN          ; COMPLETE I/O AND EXIT DRIVER

;      *************************************************************
;      *                                                           *
```

```
;       *       BUFFER WAS LEGAL, CONVERT VIRTUAL ADDRESS TO        *
;       *       ADDRESS DOUBLEWORD AND STORE PARAMETERS             *
;       *                                                           *
;       *************************************************************


;       +-----------------------------------------------------------+
;       |                                                           |
;       |       THE INPUT PARAMETERS FOR $RELOC ARE:                |
;       |                                                           |
;       |       R0 = USER VIRTUAL ADDRESS TO RELOCATE               |
;       |                                                           |
;       |       THE OUTPUT PARAMETERS ARE:                          |
;       |                                                           |
;       |       R1 = APR6 RELOCATION BIAS OF USER BUFFER            |
;       |       R2 = DISPLACEMENT IN BLOCK + 140000                 |
;       |                                                           |
;       +-----------------------------------------------------------+

10$:     CALL    $RELOC             ; RELOCATE BUFFER ADDRESS
         MOV     R1,I.PRM+10(R3)    ; SAVE APR BIAS OF SOURCE BUFFER
         MOV     R2,I.PRM+12(R3)    ; AND DISPLACEMENT ADDRESS
         CLR     I.PRM+16(R3)       ; INDICATE NOT BUFFERED I/O

;       *************************************************************
;       *                                                           *
;       *       NOW QUEUE THE PACKET IN THE DEVICE QUEUE            *
;       *                                                           *
;       *************************************************************

         MOV     R4,R0              ; COPY POINTER TO I/O QUEUE LISTHEAD
         MOV     R3,R1              ; AND ADDRESS OF I/O PACKET

;       +-----------------------------------------------------------+
;       |                                                           |
;       |       THE INPUT PARAMETERS FOR $QINSP ARE:                |
;       |                                                           |
;       |       R0 = ADDRESS OF THE TWO WORD LISTHEAD               |
;       |       R1 = ADDRESS OF THE PACKET TO BE INSERTED           |
;       |                                                           |
;       |       NO OUTPUT PARAMETERS                                |
;       |                                                           |
;       +-----------------------------------------------------------+

         CALL    $QINSP             ; INSERT PACKET IN QUEUE
```

```
;       ***********************************************************
;       *                                                         *
;       *       BEGIN SERIAL PROCESSING OF I/O PACKETS            *
;       *                                                         *
;       ***********************************************************


;       +---------------------------------------------------------+
;       |                                                         |
;       |       THE INPUT PARAMETERS FOR $GTPKT ARE:              |
;       |                                                         |
;       |       R5 = ADDRESS OF THE UCB OF REQUESTING UNIT        |
;       |                                                         |
;       |       THE OUTPUT PARAMETERS ARE:                        |
;       |                                                         |
;       |       C = 0 IF A REQUEST WAS SUCCESSFULLY DEQUEUED      |
;       |               R1 = ADDRESS OF THE I/O PACKET            |
;       |               R2 = PHYSICAL UNIT NUMBER                 |
;       |               R3 = CONTROLLER INDEX                     |
;       |               R4 = SCB ADDRESS OF CONTROLLER            |
;       |               R5 = UCB ADDRESS OF UNIT                  |
;       |       C = 1 IF UNIT BUSY OR NO PACKETS QUEUED           |
;       |                                                         |
;       +---------------------------------------------------------+

BMIN1:  CALL    $GTPKT          ; ATTEMPT TO GET A REQUEST
        BCC     20$             ; IF CC WE GOT ONE
        RETURN                  ; DEVICE BUSY OR QUEUE EMPTY
20$:                            ; REFERENCE LABEL
;       ***********************************************************
;       *                                                         *
;       *       ATTEMPT TO ALLOCATE CLOCK BLOCK                   *
;       *                                                         *
;       ***********************************************************
;

        MOV     R1,R3           ; COPY I/O PACKET ADDRESS
        MOV     #C.LGTH,R1      ; SET LENGTH OF CLOCK BLOCK

;       +---------------------------------------------------------+
;       |                                                         |
;       |       THE INPUT PARAMETERS FOR $ALOCB ARE:              |
;       |                                                         |
;       |       R1 = SIZE OF THE BLOCK TO ALLOCATE (IN BYTES)     |
;       |                                                         |
;       |       THE OUTPUT PARAMETERS ARE:                        |
;       |                                                         |
;       |       C = 0 IF A BLOCK WAS SUCCESSFULLY ALLOCATED       |
;       |               R0 = ADDRESS OF THE ALLOCATED BLOCK       |
;       |               R1 = LENGTH OF THE ALLOCATED BLOCK        |
;       |       C = 1 IF NO BLOCK IS CURRENTLY AVAILABLE          |
```

```
;      |                                                           |
;      +-----------------------------------------------------------+

        CALL    $ALOCB          ; ATTEMPT TO ALLOCATE
        BCC     30$             ; IF CC SUCCESSFUL
        MOV     #IE.NOD&377,R0  ; SET I/O STATUS
;      +-----------------------------------------------------------+
;      |                                                           |
;      |       THE INPUT PARAMETERS FOR $IOALT ARE:                |
;      |                                                           |
;      |       R0 = FIRST  WORD OF I/O STATUS BLOCK                |
;      |       R1 = SECOND WORD OF I/O STATUS BLOCK                |
;      |       R2 = STARTING AND FINAL RETRY COUNTS                |
;      |            (IF AN ERROR LOGGING DEVICE)                   |
;      |       R5 = UCB ADDRESS OF UNIT TO COMPLETE                |
;      |                                                           |
;      |       THE OUTPUT PARAMETERS ARE:                          |
;      |                                                           |
;      |       R4 IS DESTROYED                                     |
;      |                                                           |
;      +-----------------------------------------------------------+

        CALL    $IOALT          ; AND COMPLETE THE I/O
        BR      BMIN1           ; GO LOOK FOR MORE WORK
30$:    MOV     R0,I.PRM+14(R3) ; SAVE ADDRESS OF CLOCK BLOCK

;      *************************************************************
;      *                                                           *
;      *       DETERMINE IF I/O REQUEST IS BUFFERABLE              *
;      *                                                           *
;      *************************************************************


;      +-----------------------------------------------------------+
;      |                                                           |
;      |       THE INPUT PARAMETERS FOR $TSTBF ARE:                |
;      |                                                           |
;      |       R3 = ADDRESS OF I/O PACKET TO TEST                  |
;      |                                                           |
;      |       THE OUTPUT PARAMETERS ARE:                          |
;      |                                                           |
;      |       C = 0 IF REQUEST MAY BE BUFFERED                    |
;      |       C = 1 IF REQUEST MAY NOT BE BUFFERED                |
;      |                                                           |
;      +-----------------------------------------------------------+
;

        CALL    $TSTBF          ; TEST FOR BUFFERABLE I/O REQUEST
        BCS     40$             ; IF CS CAN'T ALLOCATE A BUFFER
```

```
;       *************************************************************
;       *                                                           *
;       *       ATTEMPT TO ALLOCATE A BUFFER                        *
;       *                                                           *
;       *************************************************************
;

;         MOV     I.PRM+4(R3),R1  ; GET LENGTH OF BUFFER
;         CMP     R1,#BUFLIM      ; BIGGER THAN BUFFER LIMIT ?
;         BHI     40$             ; IF HI YES, DON'T BUFFER
;

;       +-----------------------------------------------------------+
;       |                                                           |
;       |       THE INPUT PARAMETERS FOR $ALOCB ARE:                |
;       |                                                           |
;       |       R1 = SIZE OF THE BLOCK TO ALLOCATE (IN BYTES)       |
;       |                                                           |
;       |       THE OUTPUT PARAMETERS ARE:                          |
;       |                                                           |
;       |       C = 0 IF A BLOCK WAS SUCCESSFULLY ALLOCATED         |
;       |               R0 = ADDRESS OF THE ALLOCATED BLOCK         |
;       |               R1 = LENGTH OF THE ALLOCATED BLOCK          |
;       |       C = 1 IF NO BLOCK ISCURRENTLY AVAILABLE             |
;       |                                                           |
;       +-----------------------------------------------------------+
;

         CALL    $ALOCB           ; TRY TO ALLOCATE BUFFER
         BCS     40$              ; IF CS COULDN'T GET ONE

;       *************************************************************
;       *                                                           *
;       *       COPY USER BUFFER TO INTERNAL BUFFER                 *
;       *                                                           *
;       *************************************************************
;

         MOV     R0,R4            ; SET ADDRESS OF DESTINATION BUFFER
         MOV     R3,R5            ; SAVE ADDRESS OF I/O PACKET
         MOV     I.PRM+4(R5),R0   ; SET LENGTH OF TRANSFER
         MOV     I.PRM+10(R5),R1  ; SET BIAS OF SOURCE BUFFER
         MOV     I.PRM+12(R5),R2  ; AND DISPLACEMENT
         BIC     #140000,R2       ; STRIP OFF APR6 ADDRESS BITS
         BIS     #120000,R2       ; AND SUBSTITUTE APR5
         MOV     R4,I.PRM+10(R5)  ; SET INTERNAL BUFFER ADDRESS INTO
                                  ; PACKET

;       +-----------------------------------------------------------+
;       |                                                           |
```

DRIVER CODE

```
;      |      THE INPUT PARAMETERS FOR $BLXIO ARE:                    |
;      |                                                              |
;      |      R0 = NUMBER OF BYTES TO MOVE                            |
;      |      R1 = SOURCE APR 5 BIAS                                  |
;      |      R2 = SOURCE DISPLACEMENT                                |
;      |      R3 = DESTINATION APR6 BIAS                              |
;      |      R4 = DESTINATION DISPLACEMENT                           |
;      |                                                              |
;      |      THE OUTPUT PARAMETERS ARE:                              |
;      |                                                              |
;      |      R0 ALTERED                                              |
;      |      R1,R3 PRESERVED                                         |
;      |      R2,R4 POINT TO LAST BYTE OF SOURCE/DESTINATION +1       |
;      |                                                              |
;      +--------------------------------------------------------------+
;

        CALL    $BLXIO          ; COPY TO INTERNAL BUFFER

;      ***************************************************************
;      *                                                             *
;      *      CONVERT TO BUFFERED I/O REQUEST                        *
;      *                                                             *
;      ***************************************************************
;

        MOV     R5,R3           ; COPY I/O PACKET ADDRESS BACK

;      +--------------------------------------------------------------+
;      |                                                              |
;      |      THE INPUT PARAMETERS FOR $INIBF ARE:                    |
;      |                                                              |
;      |      R3 = ADDRESS OF THE I/O PACKET TO BUFFER                |
;      |                                                              |
;      |      NO OUTPUT PARAMETERS.                                   |
;      |                                                              |
;      +--------------------------------------------------------------+
;

        CALL    $INIBF          ; INITIALIZE BUFFERED I/O

;      ***************************************************************
;      *                                                             *
;      *      QUEUE THE CLOCK BLOCK                                  *
;      *                                                             *
;      ***************************************************************
;
```

```
40$:    MOV     I.PRM+14(R3),R0 ; GET ADDRESS OF CLOCK BLOCK
        MOV     #CLKSRV,C.SUB(R0) ; SET ADDRESS OF SUBROUTINE
        CLR     R1              ; HIGH ORDER DELTA TIME
        MOV     I.PRM+6(R3),R2  ; LOW ORDER PART
        MOV     #C.SYST,R4      ; SET REQUEST TYPE
        MOV     R3,R5           ; USE PACKET ADDRESS AS IDENTIFIER


;       +------------------------------------------------------------+
;       |                                                            |
;       |       THE INPUT PARAMETERS FOR $CLINS ARE:                 |
;       |                                                            |
;       |       R0 = ADDRESS OF THE CLOCK BLOCK TO QUEUE             |
;       |       R1 = HIGH ORDER HALF OF DELTA TIME                   |
;       |       R2 = LOW ORDER HALF OF DELTA TIME                    |
;       |       R4 = REQUEST TYPE                                    |
;       |       R5 = ADDRESS OF REQUESTING TASK OR IDENTIFIER        |
;       |                                                            |
;       |       NO OUTPUT PARAMETERS.                                |
;       |                                                            |
;       +------------------------------------------------------------+
;

        CALLR   $CLINS          ; QUEUE CLOCK BLOCK AND TEMPORARILY
                                ; EXIT THE DRIVER


;       *************************************************************
;       *                                                           *
;       *       C L O C K   E N T R Y   P O I N T                   *
;       *                                                           *
;       *************************************************************
;

;       *************************************************************
;       *                                                           *
;       *       CHECK TO SEE IF THE I/O WAS BUFFERED                *
;       *                                                           *
;       *************************************************************
;

CLKSRV: MOV     C.TCB(R4),R5    ; GET ADDRESS OF I/O PACKET
        TST     I.PRM+16(R5)    ; WAS IT BUFFERED I/O
        BNE     50$             ; IF NE YES, GO QUEUE KERNEL AST


;       *************************************************************
;       *                                                           *
;       *       COULDN'T BUFFER, PERFORM COPY HERE AND NOW          *
;       *                                                           *
;       *************************************************************
```

```
        MOV     I.PRM+4(R5),R0  ; SET LENGTH TO TRANSFER
        MOV     I.PRM+10(R5),R1 ; BIAS OF SOURCE BUFFER
        MOV     I.PRM+12(R5),R2 ; DISPLEACEMENT OF SOURCE
        BIC     #140000,R2      ; STRIP OFF APR6 ADDRESS BITS
        BIS     #120000,R2      ; AND CONVERT TO APR5
        MOV     I.PRM(R5),R3    ; SET BIAS OF DESTINATION
        MOV     I.PRM+2(R5),R4  ; SET DISPLACEMENT

;       +-------------------------------------------------------------+
;       |                                                             |
;       |       THE INPUT PARAMETERS FOR $BLXIO ARE:                  |
;       |                                                             |
;       |       R0 = NUMBER OF BYTES TO MOVE                          |
;       |       R1 = SOURCE APR5 BIAS                                 |
;       |       R2 = SOURCE DISPLACEMENT                              |
;       |       R3 = DESTINATION APR6 BIAS                            |
;       |       R4 = DESTINATION DISPLACEMENT                         |
;       |                                                             |
;       |       THE OUTPUT PARAMETERS ARE:                            |
;       |                                                             |
;       |       R0 ALTERED                                            |
;       |       R1,R3 PRESERVED                                       |
;       |       R2,R4 POINT TO LAST BYTE OF SOURCE/DESTINATION +1     |
;       |                                                             |
;       +-------------------------------------------------------------+
;


        CALL    $BLXIO          ; COPY BUFFER
        MOV     I.PRM+14(R5),R0 ; GET ADDRESS OF CLOCK BLOCK
        MOV     #C.LGTH,R1      ; GET LENGTH OF CLOCK BLOCK
;       +-------------------------------------------------------------+
;       |                                                             |
;       |       THE INPUT PARAMETERS FOR $DEACB ARE:                  |
;       |                                                             |
;       |       R0 = ADDRESS OF BLOCK TO DEALLOCATE                   |
;       |       R1 = LENGTH OF BLOCK TO DEALLOCATE                    |
;       |                                                             |
;       |       NO OUTPUT PARAMETERS.                                 |
;       |                                                             |
;       +-------------------------------------------------------------+
;

        CALL    $DEACB          ; DEALLOCATE IT
        MOV     R5,R3           ; COPY PACKET ADDRESS FOR $IODON
BMSUC:  MOV     #IS.SUC&377,R0  ; SET FINAL I/O STATUS
        MOV     I.PRM+4(R3),R1  ; AND LENGTH OF TRANSFER = REQUESTED
```

DRIVER CODE

```
BMDON:  MOV     I.UCB(R3),R5     ; GET UCB ADDRESS OF DEVICE

;       +------------------------------------------------------------+
;       |                                                            |
;       |       THE INPUT PARAMETERS FOR $IODON ARE:                 |
;       |                                                            |
;       |       R0 = FIRST  WORD OF I/O STATUS BLOCK                 |
;       |       R1 = SECOND WORD OF I/O STATUS BLOCK                 |
;       |       R2 = STARTING AND FINAL RETRY COUNTS                 |
;       |            (IF AN ERROR LOGGING DEVICE)                    |
;       |       R5 = UCB ADDRESS OF UNIT TO COMPLETE                 |
;       |                                                            |
;       |       THE OUTPUT PARAMETERS ARE:                           |
;       |                                                            |
;       |       R4 IS DESTROYED                                      |
;       |                                                            |
;       +------------------------------------------------------------+
;
        CALL    $IODON           ; COMPLETE THE I/O
        BR      BMIN1            ; GO LOOK FOR MORE WORK
;       **************************************************************
;       *                                                            *
;       *       BUFFERED I/O, CONVERT I/O PACKET TO KERNEL           *
;       *       AST AND EXIT FROM DRIVER                             *
;       *                                                            *
;       **************************************************************

50$:    MOV     R4,R3            ; COPY CLOCK BLOCK ADDRESS FOR $REQUE
        MOV     I.TCB(R5),R0     ; POINT TO TCB OF TASK
        TST     (R4)+            ; SKIP LINK WORD
        MOV     #AK.GBI,(R4)+    ; SET A.CBL=AK.GBI
        MOV     KISAR5,(R4)+     ; SET APR BIAS OF SERVICE ROUTINE
        MOV     #KATSRV,(R4)+    ; SET ADDRESS OF PROCESSING ROUTINE
        MOV     R5,(R4)+         ; SAVE I/O PACKET ADDRESS IN CLOCK BLOCK

;       +------------------------------------------------------------+
;       |                                                            |
;       |       THE INPUT PARAMETERS FOR $REQUE ARE:                 |
;       |                                                            |
;       |       R0 = TCB ADDRESS TO QUEUE AST BLOCK TO               |
;       |       R3 = ADDRESS OF THE PACKET TO QUEUE                  |
;       |                                                            |
;       |       NO OUTPUT PARAMETERS.                                |
;       |                                                            |
;       +------------------------------------------------------------+

        CALLR   $REQUE           ; QUEUE AST TO TASK

;       **************************************************************
;       *                                                            *
;       *       K E R N E L   A S T   E N T R Y   P O I N T          *
```

```
;       *                                                           *
;       *************************************************************

;       *************************************************************
;       *                                                           *
;       *       GET PCB ADDRESS AND SEE IF PARTITION IS RESIDENT    *
;       *                                                           *
;       *************************************************************

KATSRV: MOV     10(R3),R5       ; GET I/O PACKET ADDRESS
        MOV     I.PRM+16(R5),R1 ; GET PCB ADDRESS OF BUFFER REGION
        BEQ     70$             ; IF EQ THERE IS NO COPY TO PERFORM

;       +-----------------------------------------------------------+
;       |                                                           |
;       |       THE INPUT PARAMETERS FOR $TSPAR ARE:                |
;       |                                                           |
;       |       R0 = ADDRESS OF THE PACKET (THE KERNEL AST BLOCK)   |
;       |       R1 = PCB ADDRESS OF THE PCB CONTAINING THE BUFFER   |
;       |       R5 = TCB ADDRESS OF ASSOCIATED TASK                 |
;       |                                                           |
;       |       THE OUTPUT PARAMETERS ARE                           |
;       |                                                           |
;       |       C = 0 IF REGION IS RESIDENT AND CAN BE ACCESSED     |
;       |       C = 1 IF REGION IS NOT RESIDENT AND AST HAS         |
;       |            BEEN QUEUED                                    |
;       |                                                           |
;       +-----------------------------------------------------------+
;

        CALL    $TSPAR          ; REGION IN MEMORY ?
        BCC     60$             ; IF CC REGION IN MEMORY
        RETURN                  ; ELSE PARTITION AST HAS BEEN QUEUED

;       *************************************************************
;       *                                                           *
;       *       PERFORM BUFFER COPY OPERATION                       *
;       *                                                           *
;       *************************************************************
;

60$:    MOV     I.PRM+4(R5),R0  ; GET COUNT OF BYTES
        MOV     I.PRM+10(R5),R2 ; SET SOURCE BUFFER ADDRESS
        MOV     P.REL(R1),R3    ; GET STARTING BIAS OF PARTITION
        ADD     I.PRM(R5),R3    ; AND ADD IN OFFSET
        MOV     I.PRM+2(R5),R4  ; SET DISPLACEMENT

;       +-----------------------------------------------------------+
;       |                                                           |
;       |       THE INPUT PARAMETERS FOR $BLXIO ARE:                |
;       |                                                           |
```

```
;       |       R0 = NUMBER OF BYTES TO MOVE                          |
;       |       R1 = SOURCE APR 5 BIAS                                |
;       |       R2 = SOURCE DISPLACEMENT                              |
;       |       R3 = DESTINATION APR6 BIAS                            |
;       |       R4 = DESTINATION DISPLACEMENT                         |
;       |                                                             |
;       |       THE OUTPUT PARAMETERS ARE                             |
;       |                                                             |
;       |       R0 ALTERED                                            |
;       |       R1,R3 PRESERVED                                       |
;       |       R2,R4 POINT TO LAST BYTE OF SOURCE/DESTINATION +1     |
;       |                                                             |
;       +-------------------------------------------------------------+
;

        CALL    $BLXIO          ; COPY THE BUFFER
        MOV     I.PRM+10(R5),R0 ; GET BUFFER ADDRESS AGAIN
        MOV     I.PRM+4(R5),R1  ; GET LENGTH OF BUFFER

;       +-------------------------------------------------------------+
;       |                                                             |
;       |       THE INPUT PARAMETERS FOR $DEACB ARE:                  |
;       |                                                             |
;       |       R0 = ADDRESS OF BLOCK TO DEALLOCATE                   |
;       |       R1 = LENGTH OF BLOCK TO DEALLOCATE                    |
;       |                                                             |
;       |       NO OUTPUT PARAMETERS.                                 |
;       |                                                             |
;       +-------------------------------------------------------------+

        CALL    $DEACB          ; DEALLOCATE IT

;       ***************************************************************
;       *                                                             *
;       *       IF THIS WASN'T A REGION LOAD AST, FINISH THE I/O      *
;       *                                                             *
;       ***************************************************************

70$:    MOV     I.PRM+14(R5),R0 ; RETRIEVE AST BLOCK ADDRESS
        TST     (R0)            ; WAS THIS A REGION LOAD AST ?
        BNE     80$             ; IF NE YES

        MOV     #C.LGTH,R1      ; SET LENGTH OF CLOCK BLOCK

;       +-------------------------------------------------------------+
;       |                                                             |
;       |       THE INPUT PARAMETERS FOR $DEACB ARE:                  |
;       |                                                             |
;       |       R0 = ADDRESS OF BLOCK TO DEALLOCATE                   |
;       |       R1 = LENGTH OF BLOCK TO DEALLOCATE                    |
;       |                                                             |
```

```
;       |       NO OUTPUT PARAMETERS.                          |
;       |                                                      |
;       +------------------------------------------------------+

         CALL    $DEACB              ; DEALLOCATE CLOCK BLOCK
         MOV     I.IOSB(R5),R3       ; GET VIRTUAL ADDRESS OF I/O STATUS
                                     ; BLOCK
         MOV     #IS.SUC&377,-(SP) ; SET FIRST I/O STATUS WORD
         MTPD$   (R3)+               ; WRITE FIRST WORD OF STATUS (MAY
                                     ; TRAP)
         MOV     I.PRM+4(R5),-(SP) ; SET SECOND WORD OF I/O STATUS
         MTPD$   (R3)                ; WRITE SECOND WORD (MAY TRAP)
         CLR     I.IOSB(R5)          ; PREVENT $IODON ATTEMPT TO WRITE
                                     ; STATUS
         MOV     R5,R3               ; COPY I/O PACKET ADDRESS
         JMP     BMSUC               ; FINISH IN COMMON CODE

;       ********************************************************
;       *                                                      *
;       *       RECONVERT REGION LOAD AST TO A TASK AST        *
;       *                                                      *
;       ********************************************************
;
80$:     MOV     R0,R3               ; COPY BLOCK ADDRESS
         CLR     10(R0)              ; INDICATE NO BUFFER NEXT TIME
         MOV     I.TCB(R5),R0        ; GET TCB ADDRESS

;       +------------------------------------------------------+
;       |                                                      |
;       |       THE INPUT PARAMETERS FOR $REQUE ARE:           |
;       |                                                      |
;       |       R0 = TCB ADDRESS TO QUEUE AST BLOCK TO         |
;       |       R3 = ADDRESS OF THE PACKET TO QUEUE            |
;       |                                                      |
;       |       NO OUTPUT PARAMETERS.                          |
;       |                                                      |
;       +------------------------------------------------------+

         CALLR   $REQUE              ; REQUEUE TASK AST AND EXIT AST
                                     ; SERVICE

;       ********************************************************
;       *                                                      *
;       *       MISCELLANEOUS ENTRY POINTS                     *
;       *                                                      *
;       ********************************************************

;       ********************************************************
;       *                                                      *
;       *       C A N C E L   E N T R Y   P O I N T            *
;       *                                                      *
```

```
;       *       WE COULD DEQUEUE PENDING CLOCK REQUEST, ETC HERE,   *
;       *       BUT WE DON'T, WE JUST LET THEM COMPLETE LATER       *
;       *                                                           *
;       *************************************************************

BMCAN:

;       *************************************************************
;       *                                                           *
;       *       T I M E O U T   E N T R Y   P O I N T               *
;       *                                                           *
;       *       SINCE THERE'S NO PHYSICAL DEVICE TO TIME OUT, NO-OP *
;       *                                                           *
;       *************************************************************

BMOUT:

;       *************************************************************
;       *                                                           *
;       *       P O W E R F A I L   E N T R Y   P O I N T           *
;       *                                                           *
;       *       POWERFAIL DOESN'T AFFECT NON-EXISTENT DEVICES       *
;       *                                                           *
;       *************************************************************

BMPWF:

;       *************************************************************
;       *                                                           *
;       *       S T A T U S   C H A N G E   E N T R Y   P O I N T S *
;       *                                                           *
;       *       DON'T NEED TO TOUCH NON-EXISTENT DEVICE, JUST LET   *
;       *       EXEC PUT DEVICE ON/OFF LINE                         *
;       *                                                           *
;       *************************************************************

BMKRB:
BMUCB:
        RETURN                  ; ALL THESE ARE NO-OP FOR NOW

        .END
```

CHAPTER 9

ACCESSING VIDEO HARDWARE AND TERMINAL SUBSYSTEM

This chapter provides reference information for programmers needing access to the Professional's video hardware, as well as to the text-handling component of the Terminal Subsystem. (For information on the graphics capabilities, see the *PRO/GIDIS Manual*.)

To use the information provided here, you should be familiar with the Professional hardware at the level of detail provided in the *Professional 300 Series Technical Manual*. Also, your familiarity with P/OS should include an understanding of the Executive directives and the Application Task Builder. This chapter does not include step-by-step instruction.

9.1 APPLICATION LEVEL ACCESS TO THE VIDEO HARDWARE

Under P/OS, the video generator device registers and display memory are generally accessed only by the Terminal Subsystem, with a higher-level interface provided for applications. However, it is also possible for an application to directly access the hardware.

The main reasons for having an application access the hardware directly are that the application might be able to:

- Achieve faster throughput than if it went through the supported system services.
- Provide functionality that the system software doesn't support.

The main reasons for NOT having an application access the hardware directly are as follows:

- Future versions of the system software may interact differently with the video hardware. An application that accesses the hardware directly may not work under all versions of P/OS. DIGITAL is not and will not attempt to achieve this type of compatibility.

- The video hardware may change. The system software keeps up with such changes, but if the application accesses the hardware directly (thus does not use the system services), it will also have to be adapted to account for the video hardware changes.
- Developing an application that accesses the hardware directly requires considerable familiarity with the hardware. The relevant sections of the *Professional 300 Series Technical Manual*, along with the contents of this document, are pre-requisites to acquiring this familiarity, but are not necessarily sufficient by themselves.

This document deals only with a Professional running P/OS, but some of the information may be useful for applications running under other operating systems.

9.1.1 Disabling the Terminal Subsystem

Before directly accessing the video hardware, an application must ensure that the Terminal Subsystem is not active. Failure to disable the Terminal Subsystem can have unpredictable results, including system software failure.

Steps for disabling the Terminal Subsystem (P/OS Versions 1.7 and later) are as follows.

1. Send a reset-to-initial-state (RIS) sequence, <ESC>c. This resets the Terminal Subsystem to its initial state and clears the screen.
2. Send a disable cursor sequence, <ESC>[?25l. This turns the text mode cursor off.

After disabling the cursor, the Terminal Subsystem accesses the video hardware only when it is requested to display something, or when it blanks the screen at the end of its time-out period.

If the application combines requests to the Terminal Subsystem with its own manipulation of the video hardware, it must save and restore the contents of the following device registers:

- Control and Status Register

- Plane 1 Control Register
- Plane 2 and 3 Control Register
- Memory Base Register (should not be modified at all)

CAUTION

DO NOT SET THE INTERRUPT ENABLE BITS

The results of doing so are unpredictable, and probably undesirable. The system could hang or crash.

9.1.2 Accessing the Video Device Registers

In the CTI architecture, as on other PDP-11 buses, devices are controlled via device registers, which appear as memory in the top 8KB (the I/O page) of the bus address space.

However, unlike those on UNIBUS and Q-BUS devices, the device registers on a CTI device do not have fixed addresses on the bus. Instead, each option slot has a 128-byte device register address space, the location of which can be found in the Professional 300 Series Technical Manual.

The registers on a given module appear at a fixed displacement within the address space of the slot containing the module. This means that in order to access the device registers on the video, software must determine at run-time which slot it is in. The WIMP$ Executive Directive provides the means for doing this.

The WIMP$ Directive returns a dump of the configuration table, including the IDs of the devices in all the slots. Scan the list to find out which slot contains the video controller. The *P/OS System Reference Manual* lists the device IDs.

NOTE

The ID value for the PC350 video controller is different from that of the PC380 video controller. Be sure to use the correct ID.

The slot number gives you the bus address. You can then taskbuild your application in a resident common in partition CTPAGE, which covers the I/O page.

9.1.3 Accessing Video Memory Through the Bus

Because the Terminal Subsystem puts the video display memory on the CTI bus at boot time, the CPU can read from and write to this memory without using the device registers. P/OS uses this capability.

The video memory occupies a partition called BITMAP, and a region called TFWBMP fills this partition. The Terminal Subsystem creates TFWBMP and places it on the bus at boot time. An application can attach the region and map a portion of it.

TFWBMP is 32KB (PC350) or 128KB (PC380). The displayed portion of TFWBMP is the first 30KB (PC350, or PC380 low resolution) or 60KB (PC380 high resolution). A bit in the CSR indicates the resolution, as described in the *Technical Manual*.

Note that the least significant portion of the first word of the region corresponds to the upper left corner of the screen. If there is an Extended Bit Option (EBO) in the system, all three planes of memory share the same bus address space. (To determine if a system has an EBO option, check the CSR of the video hardware.)

To read from or write to any of the planes (whether or not there is an EBO), the memory reference enable bit (bit 5) in its plane control register must be set to 1. If there is an EBO, the memory reference enable bit can be set for more than one plane.

Reads will come in ascending order (1, 2, 3) from each plane that has the memory reference enable bit set. Plane numbers are defined by the hardware and do not necessarily correspond to any software numbering scheme. Writes will go to all planes that have the memory reference enable bit set.

Remember that a transfer, once started, can take a long time. Check the Done bit before modifying any registers other than the X and Y registers, unless you are certain that the transfer is complete.

9.1.4 The Screen Timer

A screen time-out feature is built into the Terminal Subsystem to prevent the burn-in of a static image in the screen phosphor. The Terminal Subsystem blanks the screen if there has been no keyboard or video activity for 30 minutes. This is accomplished by setting the horizontal resolution to off in the Plane 1 Control Register.

For applications that process input from the keyboard, the screen timer is not a problem, but for display-only applications it can be a problem because the Terminal Subsystem cannot detect the ongoing video activity.

One solution to this problem is to send a null byte to the screen (via the Terminal Subsystem) at regular intervals (at least every 30 minutes). This will cause the Terminal Subsystem to reset its timer, without causing any visual effects.

9.1.5 Returning the Video Hardware to the System

After the application has completed, it must restore the hardware device registers, especially the Plane 1 control register. Failure to do so may cause the system to crash during a split-screen scroll or insert/delete operation. You should re-initialize the Terminal Subsystem by issuing a RIS sequence.

APPENDIX A

P/OS SYSTEM DATA STRUCTURES AND SYMBOLIC DEFINITIONS

This appendix describes the P/OS system macros that supply symbolic offsets for data structures listed in Table A-1.

The data structures are defined by macros in the Executive macro library. To reference any of the data structure offsets from your code, include the macro name in an .MCALL directive and invoke the macro. For example:

```
        .MCALL DCBDF$            ;Define DCB offsets
```

NOTE

> All physical offsets and bit definitions are subject to change in future releases of the operating system. Code that accesses system data structures should always use the symbolic offsets rather than the physical offsets.

The first two arguments, <:> and <=>, make all definitions global. If they are left blank, the definitions will be local. The SYSDEF argument causes the variable part of a data structure to be defined.

All of these macros are in the Executive macro library, LB:[1,5]EXEMC.MLB. All except F11DF$ and ITBDF$ are also in the Executive definition library, LB:[1,1]EXELIB.OLB.

Table A-1: Summary of System Data Structure Macros

Macro	Arguments	Data Structures
ABODF$		Task abort and termination notification message codes
ACNDF$		UAB definitions
ACTDF$		Account file definitions
BCKDF$		Bugcheck code
CLKDF$	<:>,<=>	Clock queue control block
CTBDF$	<:>,<=>	Controller table
DCBDF$	<:>,<=>,SYSDEF	Device control block
DDT$		Macro to generate driver dispatch table
F11DF$	<:>,<=>,SYSDEF	Files-11 data structures (volume control block, mount list entry, file control block, file window block, locked block list node)
GTPKT$		Macro to generate $GTPKT UCB address
HDRDF$	<:>,<=>	Task header and window block
HWDDF$	<:>,<=>,SYSDEF	Hardware register addresses and feature mask definitions
INTSV$		Macro to generate interrupt entry code to determine UCB address
ITBDF$	<:>,<=>,SYSDEF	Interrupt transfer block
KRBDF$	<:>,<=>	Controller request block
LBLDF$		Task image file label block definitions
LNMDF$		Logical name block (LNB)

PCBDF$	<:>,<=>,SYSDEF	Partition control block and attachment descriptor
PKTDF$	<:>,<=>	I/O packet, AST control block, offspring control block, group global event flag control block, and CLI parser block
QIOSY$		I/O error function code definition
SCBDF$	<:>,<=>,SYSDEF	Status control block
TCBDF$	<:>,<=>,SYSDEF	Task control block
TTSYM$		Terminal driver symbols
UCBDF$	<:>,<=>,TTDEF,SYSDEF	Unit control block

A.1 ABODF$

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

177774   S.CACT=-4.        ;TASK STILL ACTIVE
177776   S.CEXT=-2.        ;TASK EXITED NORMALLY
000000   S.COAD=0.         ;ODD ADDRESS AND TRAPS TO 4
000002   S.CSGF=2.         ;SEGMENT FAULT
000004   S.CBPT=4.         ;BREAK POINT OR TRACE TRAP
000006   S.CIOT=6.         ;IOT INSTRUCTION
000010   S.CILI=8.         ;ILLEGAL OR RESERVED INSTRUCTION
000012   S.CEMT=10.        ;NON RSX EMT INSTRUCTION
000014   S.CTRP=12.        ;TRAP INSTRUCTION
000016   S.CFLT=14.        ;11/40 FLOATING POINT EXCEPTION
000020   S.CSST=16.        ;SST ABORT-BAD STACK
000022   S.CAST=18.        ;AST ABORT-BAD STACK
000024   S.CABO=20.        ;ABORT VIA DIRECTIVE
000026   S.CLRF=22.        ;TASK LOAD REQUEST FAILURE
000030   S.CCRF=24.        ;TASK CHECKPOINT READ FAILURE
000032   S.IOMG=26.        ;TASK EXIT WITH OUTSTANDING I/O
000034   S.PRTY=28.        ;TASK MEMORY PARITY ERROR
000036   S.CPMD=30.        ;TASK ABORTED WITH PMD REQUEST
000040   S.CELV=32.        ;TI: VIRTUAL TERMINAL WAS ELIMINATED
000042   S.CINS=34.        ;TASK INSTALLED IN 2 DIFFERENT SYSTEMS
000044   S.CAFF=36.        ;TASK ABORTED DUE TO BAD AFFINITY (REQUIRED BUS
                           ;RUNS ARE OFFLINE OR NOT PRESENT)
000046   S.CCSM=38.        ;BAD CSM PARAMETERS OR BAD STACK
000050   S.COTL=40.        ;TASK HAS RUN OVER ITS TIME LIMIT
000052   S.CTKN=42.        ;ABORT VIA DIRECTIVE WITH NO TKTN MESSAGE

         ;
         ; TASK TERMINATION NOTIFICATION MESSAGE CODES
         ;

000000   T.NDNR=0          ;DEVICE NOT READY
000002   T.NDSE=2          ;DEVICE SELECT ERROR
000004   T.NCWF=4          ;CHECKPOINT WRITE FAILURE
000006   T.NCRE=6          ;CARD READER HARDWARE ERROR
000010   T.NDMO=8.         ;DISMOUNT COMPLETE
000012   T.NUER=10.        ;UNRECOVERABLE ERROR
000014   T.NLDN=12.        ;LINK DOWN (NETWORKS)
000016   T.NLUP=14.        ;LINK UP (NETWORKS)
000020   T.NCFI=16.        ;CHECKPOINT FILE INACTIVE
000022   T.NUDE=18.        ;UNRECOVERABLE DEVICE ERROR
000024   T.NMPE=20.        ;MEMORY PARITY ERROR
000026   T.NKLF=22.        ;UCODE LOADER NOT INSTALLED
000030   T.NAAF=24.        ;ACCOUNTING ALLOCATION FAILURE
```

ABODF$

```
000032    T.NTAF=26.       ;ACCOUTING TAB ALLOCATION FAILURE
000034    T.NDEB=28.       ;TASK HAS NO DEBUGGING AID
000036    T.NRCT=30.       ;REPLACEMENT CONTROL TASK NOT INSTALLED
000040    T.NWBL=32.       ;WRITE BACK CACHING DATA LOST. UNIT WRITE
                          ;LOCKED
000042    T.NVER=34.       ;MOUNT VERIFICATION TASK NOT INSTALLED
```

A.2 ACNDF$

```
;+
; ACCOUNTING  BLOCK OFFSET AND STATUS DEFINITIONS
; FOR EACH TRANSACTION TYPE.
;
;
; HEADER COMMON TO ALL TRANSACTIONS
;-

000000          .ASECT
        000000  .=0

000000  B.LNK:.BLKW     1       ;LINK TO NEXT IN SYSLOG QUEUE
000002  B.TYP:.BLKB     1       ;TRANSACTION TYPE
000003  B.LEN:.BLKB     1       ;TRANSACTION LENGTH
000004  B.TIM:.BLKW     3       ;ENDING TIME OF TRANSACTION
000012  B.HID=.                 ;START OF HEADER IDENTIFICATION AREA
000012  B.UID:.BLKW     2       ;UNIQUE SESSION IDENT
                                ; FIRST WORD-RADIX-50, SECOND-BINARY
000016  B.ACN:.BLKW     1       ;ACCOUNT NUMBER
000020  B.TID:.BLKB     1       ;ASCII TERMINAL TYPE (V,T, OR C)
                                ;(VIRTUAL,REAL,BATCH, OR CONSOLE)
000021          .BLKB   1       ;UNIT NUMBER
        000022  B.HEND=.        ;END OF HEADER ID AREA
        000022  $$$HLN=.        ;HEADER LENGTH


;+
; ACCUMULATION FIELDS FOR TAB, UAB, AND SAB
;-

000022  B.CPU:.BLKW     2       ;TOTAL CPU TIME USED
000026  B.DIR:.BLKW     2       ;TOTAL DIRECTIVE COUNT
000032  B.QIO:.BLKW     2       ;TOTAL QIO$ COUNT
000036  B.TAS:.BLKW     2       ;TOTAL TASK COUNT
000042  B.MEM:.BLKW     3       ;RESERVED
000050  B.BEG:.BLKW     3       ;BEGINNING/LOGIN TIME
000056  B.CPUL:.BLKW    2       ;CPU LIMIT
000062  B.PNT:.BLKW     1       ;POINTER TO HIGHER LEVEL TOTALS
000064  B.STM:.BLKB     1       ;STATUS MASK
000065  $$$TLN=.                ;TOTAL'S LENGTH


;+
; USER ACCOUNT BLOCK (UAB)
;       NOTE:  UAB'S MUST END ON A WORD BOUNDARY
;-

        000065  .=$$$TLN        ;START AFTER TOTALS
000065  B.USE:.BLKB     1       ;USE COUNT
```

ACNDF$

```
000066    B.ACT:.BLKW      1       ;NUMBER OF CURRENTLY ACTIVE TASKS
000070    B.UUIC:.BLKW     1       ;LOGIN UIC
000072    B.UCB:.BLKW      1       ;POINTER TO UCB
000074    B.LGO:.BLKW      3       ;LOGOFF TIME
000102    B.ULNK:.BLKW     1       ;LINK TO NEXT UAB
000104    B.RNA:.BLKW      3       ;LOC IN SYSACCT FILE (OFFSET,VBN-HI,
                                   ;VBN-LO)
000112    B.NAM:.BLKB      14.     ;LAST NAME OF USER
000130            .BLKB    1       ;FIRST INITIAL OF USER
000131            .BLKB    1       ;FLAG BYTE FOR UAB (bs.sil) etc.
000132    B.LDS:.BLKB      10.     ;LOGIN DIRECTORY STRING
                  .IF DF R$$PRO
000144    B.CBT:.BLKW      1       ;POINTER TO CHANNEL BLOCK TABLE
                  .ENDC
          000146  B.ULEN=.         ;UAB LENGTH
          000002  $$$=     <.+77>/100     ;UAB LENGTH (ROUNDED UP TO 32
                                   ;WORD BOUND)
```

A.3 ACTDF$

```
000000          ACTDF$
000062                  .ASECT
        000000  .=0
000000          A.GRP:  .BLKB   3       ; GROUP CODE (ASCII)
000003          A.MBR:  .BLKB   3       ; MEMBER CODE
000006          A.PSWD: .BLKB   6       ; PASSWORD
000014          A.LNM:  .BLKB   14.     ; LAST NAME
000032          A.FNM:  .BLKB   12.     ; FIRST NAME
000046          A.LDAT: .BLKB   6       ; LAST LOG ON--
                                        ; DD/MM/YY HH:MM:SS
000054          A.NLOG: .BLKB   2       ; TOTAL NUMBER OF LOGONS
000056          A.SYDV: .BLKB   4       ; DEFAULT SYSTEM DEVICE
000062          A.ACN:  .BLKW   1       ;ACCOUNT NUMBER (BINARY)
000064          A.CLI:  .BLKW   2       ; RADIX-50 USER CLI
000070                  .BLKW   2       ; UNUSED
000074          A.LPRV: .BLKW   1       ;LOGIN PRIVILEGE WORD
000076          A.SID:  .BLKW   1       ; SESSION IDENTIFIER
000100          A.DDS:  .BLKB   11.     ;DEFAULT DIRECTORY STRING
000113                  .BLKB   1       ;UNUSED BYTE
000114          A.FPRO: .BLKW   1       ;DEFAULT FILE PROTECTION
000116          A.RLVL: .BLKW   1       ;ACCOUNT RECORD REV. LEVEL
        000401  AR.LVL=401
000120          A.SALT: .BLKW   1       ;16-BIT ENCRYPTION SALT VALUE
000122          A.ENCT: .BLKB   1       ;ENCRYPTION TYPE
                        ;               0 = PLAIN TEXT OR ENCRPT
                        ;               1 = PURDY-V ALGORITHM
000123                  .BLKB   1       ;UNUSED
000124          A.HPW:  .BLKW   4       ;HASHED PASSWORD
                        .IF DF R$$PRO

000134          AH.TYP: .BLKB   1       ;TYPE OF HOME DEFINITION
000135          AH.SZ1: .BLKB   1       ;IF AH.TYP=1,LENGTH OF NODENAME
                                        ;IF AH.TYP=2,LENGTH OF
                                        ;DEVICE SPEC
000136          AH.HOM:                 ;IF AH.TYP=2,START OF
                                        ;DEVICE SPEC
000136          AH.NOD: .BLKB   6       ;IF AH.TYP=1,NODENAME IN ASCII
000144          AH.SZ2: .BLKB   1       ;LENGTH OF USERNAME
000145          AH.USN: .BLKB   14.     ;USERNAME IN ASCII
000163          AH.SZ3: .BLKB   1       ;LENGTH OF PASSWORD
000164          AH.HPW: .BLKB   8.      ;HASHED PASSWORD
000174  AH.RLN=
;
;       BIT DEFINITIONS FOR AH.TYP
;
        000000  AH.NUL= 0               ;HOME VALUE IS NULL
        000001  AH.CLL= 1               ;HOME VALUE IS CLUSTER LOG TRIO
        000002  AH.DVS= 2               ;HOME VALUE IS A DEVICE SPEC
        ;               .ENDC           ;DF R$$PRO
```

ACTDF$

```
000200  A.LEN    =       128.     ; LENGTH OF CONTROL BLOCK
;
;
; BIT DEFINITIONS ON A.LPRV - LOGIN PRIVILEGE BITS
;.
000001  AL.SLV= 1                ;SLAVE TERMINAL ON LOGIN
000002  AL.DDS= 2                ;INDICATOR FOR PROLOGUE 2 FORMAT
000004  AL.SIL= 4                ;SILENT LOGIN/LOGOUT
                       .IF DF A$$LOG

        AL.AUT= 10               ;AUTO LOGIN ENABLED      ('*)
        AL.BND= 20               ;BINDING ENABLED         ('Y)
        AL.RMT= 40               ;REMOTE DIALUP 1=NO
        AL.NET= 100              ;NETWORK LOGIN 1=NO
        AL.DIS= 200              ;DISABLE THIS ACCOUNT FROM LOGIN
        AL.PRI= 400              ;PRIMARY DAYS LIMIT SET
        AL.SEC= 1000             ;SECONDARY DAYS LIMIT SET

                       .ENDC ; DF A$$LOG
;              .IF DF R$$PRO
002000  AL.NDL= 2000             ;RECORD MARKED FOR NO DELETE
004000  AL.NMD= 4000             ;RECORD CANNOT BE MODIFIED
010000  AL.NFL= 10000            ;FOREGROUND LOGIN'S DISABLED
020000  AL.FPC= 20000            ;FORCE PASSWD CHANGE
;              .ENDC  ;DF R$$PRO
```

A.4 BCKDF$

```
000100  BF.PKS=000100          ; P/OS Keyboard Handler
000200  BF.TTD=000200          ; Terminal Driver
100400  BF.PTS=100400          ; P/OS Terminal Subsystem
000300  BF.EXE=000300          ; Exec - SSTSR, General
000000  BE.IOT=000000          ; IOT in System State
000001  BE.STK=000001          ; Stack Overflow
000002  BE.BPT=000002          ; Trace Trap or Breakpoint
000003  BE.ILI=000003          ; Illegal Instruction Trap
000004  BE.ODD=000004          ; Odd Address or Other Trap 4
000005  BE.SGF=000005          ; Segment Fault
000006  BE.NPA=000006          ; A Task on P/OS Without a
                               ; Parent Aborted
000007  BE.EMT=000007          ; EMT Trap
000010  BE.TRP=000010          ; TRAP Trap
000011  BE.NOD=000011          ; Out of POOL

000400  BF.UP =000400          ; System Startup Processing
000001  BE.IN1=000001          ; Can’t Install Task CBOOT
000002  BE.SP1=000002          ; Can’t Spawn Task CBOOT
000003  BE.SP2=000003          ; Can’t Spawn Task CMAIN
000007  BE.FNF=000007          ; Required File Not Found
        ;
        ; PRO/DECnet startup failure codes
        ;
000010  BE.DSC=000010          ; DSR corrupt
000011  BE.BDP=000011          ; Bad dispatch
000012  BE.NWB=000012          ; No way to Boot via DECNA (Cluster)
000013  BE.DAF=000013          ; DSR Allocation Failure
000014  BE.VIU=000014          ; DDM Vector in use
000015  BE.NPD=000015          ; Required PDV not found.
000016  BE.NSD=000016          ; Required Hardware not present.
        ;
        ; P/OS Server network failure codes
        ;
        ;
        ; SCA facility code
        ;
000500  BF.SCA= 000500         ; SCA network
        ;
        ; SCAINT bugcheck codes
        ;
000001  BE.NCS= 000001         ; No PDV index for a required Cluster
                               ; process
000002  BE.NSP= 000002         ; Error creating Cluster secondary
                               ; pool
000003  BE.ECL= 000003         ; Error creating FS: logicals
000004  BE.SYC= 000004         ; Error creating system channel
000005  BE.SAF= 000005         ; Error allocating structures from
```

```
                                ; Cluster secondary pool
000006   BE.ASY= 000006         ; CCB allocation failure during system
                                ; channel creation
000007   BE.PAF= 000007         ; Error allocating from primary pool
000010   BE.DVL= 000010         ; Error creating device list
000011   BE.NFS= 000011         ; No FS: device
000012   BE.NCL= 000012         ; COMINI spawned SCAINT without a
                                ; command line
000013   BE.ISL= 000013         ; Command line string invalid length
         ;
         ;
         ; PPD bugcheck codes
         ;
000101   BE.ANI= 000101         ; Allocation failure during network
                                ; startup
000102   BE.PDV= 000102         ; Required PDV index not found (UNA or
                                ; EPM)
000103   BE.CTL= 000103         ; Decnet control function completed
                                ; unsuccessfully (enable port,start
                                ; channel,set protocol type)
000104   BE.UAQ= 000104         ; Entry not found in unacknowledged
                                ; queue
000105   BE.IMT= 000105         ; Illegal message type (datagram)
000106   BE.REG= 000106         ; No entry in retransmit queue
000107   BE.SB = 000107         ; System block address not in link
                                ; list
000110   BE.IDT= 000110         ; Illegal entry into dispatch table
000111   BE.CTM= 000111         ; Unack'd queue empty (CHKTMR routine)
000112   BE.XMT= 000112         ; Unack'd queue empty (XMIT routine)
000113   BE.XMC= 000113         ; Unack'd queue empty (XMITC routine)
000114   BE.VCS= 000114         ; VCSTART timed out, can't communicate
                                ; with the fileserver
         ;
         ; SCS bugcheck codes
         ;
000201   BE.ILM= 000201         ; Illegal message type (datagram)
000202   BE.SCB= 000202         ; Connect block address not in link
                                ;    list (at VCHALT processing)
000203   BE.CNT= 000203         ; Usage count not = 0 (at
                                ;    VCHALT processing)
000204   BE.DSP= 000204         ; Illegal entry into dispatch table
000205   BE.CNB= 000205         ; Connect block address not in link
                                ;   list (in DCBCT)
000206   BE.LST= 000206         ; Not enough info passed in listen vadd
000207   BE.CNN= 000207         ; Not enough info passed in connect vadd
000210   BE.ACC= 000210         ; Not enough info passed in accept vadd
000211   BE.RJC= 000211         ; Not enough info passed in reject vadd
000212   BE.XCT= 000212         ; Illegal transmission of control msg
000213   BE.SNT= 000213         ; Invalid entry into send table
         ;
         ; FSD bugcheck codes
```

```
        ;
000301  BE.NET= 000301          ; Unable to communicate with
                                ; fileserver during channel processing
        BE.DSL= 000302          ; Unable to communicate with
                                ; fileserver during device select proc
000303  BE.CHN= 000303          ; Error during channel processing, no
                                ; entry in clock block queue
000304  BE.ACO= 000304          ; Allocation failure during connection
                                ; processing
000305  BE.COF= 000305          ; Connect failed for reason other
                                ; than allocation failure
000306  BE.DEV= 000306          ; Too many devices in device list
000307  BE.ASD= 000307          ; Unable to allocate system data
                                ; structures (VCB, UCB extension)
000310  BE.NST= 000310          ; No TCB for startup task SCAINT
000311  BE.CAF= 000311          ; CCB allocation failure in retry
                                ; routine
000312  BE.SND= 000312          ; Unable to communicate with the
                                ; fileserver during wlb/wvb/ctl/ulk/rlb
000313  BE.ATT= 000313          ; Attribute list spans a CTNA buffer
000314  BE.IOP= 000314          ; I/O packet not in queue
000315  BE.NCB= 000315          ; Channel block not in link list for
                                ; delete channel
000316  BE.NCD= 000316          ; QIO to LUN without concealed device
000317  BE.NRF= 000317          ; Deaccess can't find RFCB entry in
                                ; queue
000320  BE.ILE= 000320          ; Illegal entry point in dispatch
                                ; table
000321  BE.VCH= 000321          ; VCHALT, fileserver has gone down
000322  BE.IOQ= 000322          ; Queue length is <> the number of
                                ; entries in the I/O queue
000323  BE.SYS= 000323          ; No matching system at login
000324  BE.CBF= 000324          ; Channel block table full (V3.0 only)
        ;
        ; FSI/FSA bugcheck codes
        ;
000401  BE.DBA= 000401          ; BAD DISPATCH OFFSET - FSA
000402  BE.DBI= 000402          ; BAD DISPATCH OFFSET - FSI
000403  BE.SNF= 000403          ; STRUCTURE NOT FOUND IN ACTIVE LIST
000404  BE.ENT= 000404          ; BAD ENTRY POINT
000405  BE.LDR= 000405          ; LOADR NOT FOUND IN ATL
000406  BE.VCP= 000406          ; VCP NOT ACTIVE
000407  BE.HLT= 000407          ; BAD DISPATCH FOR VCH
        ;
        ; bugcheck codes for cluster subroutines that are
        ; built with the POS exec
        ;
000501  BE.BDB= 000501          ; DLBDB. - BDB address not in active
                                ; list
```

A.5 CLKDF$

```
;+
; 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:
;-

000000   C.MRKT=0        ; MARK TIME REQUEST
000002   C.SCHD=2        ; TASK REQUEST WITH PERIODIC
                         ;  RESCHEDULING
000004   C.SSHT=4        ; SINGLE SHOT TASK REQUEST
000006   C.SYST=6        ; SINGLE SHOT INTERNAL SYSTEM SUBROUTINE
                         ;  (IDENT)
000010   C.SYTK=8        ; SINGLE SHOT INTERNAL SYSTEM SUBROUTINE
                         ;  (TASK)
000012   C.CSTP=10.      ; CLEAR STOP BIT (CONDITIONALIZED ON
                         ;  SHUFFLING)

                         ;
                         ; CLOCK QUEUE CONTROL BLOCK TYPE INDEPENDENT
                         ; OFFSET DEFINTIONS
                         ;

                 .ASECT
         .=0
000000   C.LNK:  .BLKW   1  ; CLOCK QUEUE THREAD WORD
000002   C.RQT: .BLKB    1  ; REQUEST TYPE
000003   C.EFN: .BLKB    1  ; EVENT FLAG NUMBER (MARK TIME ONLY)
000004   C.TCB: .BLKW    1  ; TCB ADDRESS OR SYSTEM SUBROUTINE
                            ;  IDENTIFICATION
000006   C.TIM: .BLKW   2   ; ABSOLUTE TIME WHEN REQUEST COMES DUE

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

000012   .=C.TIM+4               ; START OF DEPENDENT AREA
000012   C.AST:  .BLKW   1       ; AST ADDRESS
000014   C.SRC:  .BLKW   1       ; FLAG MASK WORD FOR 'BIS' SOURCE
000016   C.DST:  .BLKW   1       ; ADDRESS OF 'BIS' DESTINATION
000020           .BLKW   1       ; UNUSED

                         ;
```

CLKDF$

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

000012   .=C.TIM+4                 ; START OF DEPENDENT AREA
000012   C.RSI: .BLKW     2        ; RESCHEDULE INTERVAL IN CLOCK TICKS
000016   C.UIC: .BLKW     1        ; SCHEDULING UIC
000020   C.UAB: .BLKW     1        ; POINTER TO ASSOCIATED UAB

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

000012   .=C.TIM+4                 ; START OF DEPENDENT AREA
000012   .BLKW            2        ; TWO UNUSED WORDS
000016   .BLKW            1        ; SCHEDULING UIC
000020   .BLKW            1        ; C.UAB

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

000012   .=C.TIM+4                 ; START OF DEPENDENT AREA
000012   C.SUB:   .BLKW   1        ; SUBROUTINE ADDRESS
000014   C.AR5:   .BLKW   1        ; RELOCATION BASE (FOR LOADABLE
                                   ; DRIVERS)
000016   C.URM:   .BLKW   1        ; URM TO EXECUTE ROUTINE ON
                                   ; (MP SYSTEMS, C.SYST ONLY)
000020            .BLKW   1        ; UNUSED
000022            C.LGTH=.         ; LENGTH OF CLOCK QUEUE CONTROL
                                   ; BLOCK
000000            .PSECT
```

A.6 CTBDF$

```
;+
; CONTROLLER TABLE (CTB)
;
; THE CONTROLLER TABLE IS A CONTROL BLOCK THAT CONTAINS A VECTOR
; OF KRB ADDRESSES.  THIS VECTOR MAY BE ADDRESSED BY THE CONTROLLER
; INDEX TAKEN FROM THE INTERRUPT PS BY $INTSI.
;
;-


                  .ASECT


        .=177754

177754  L.CLK:'L' .BLKW 8.      ;START OF CLOCK BLOCK (CLK BLK IS
                                ;OPTIONAL, AND DRIVER DEPENDENT.
                                ;ALLOCATION OF THESE 8 WORDS IS NOT
                                ;REQUIRED IN THE DRIVER'S DATA BASE
                                ;UNLESS USED BY THE DRIVER ITSELF)
177774  L.DID:'L' .BLKW 1       ;HARDWARE DEVICE ID (WORD ALLOCATION
                                ;ALWAYS REQUIRED. MAY BE 0).
177776  L.ICB:'L' .BLKW 1       ;ICB CHAIN FOR THIS CTB
000000  L.LNK:'L' .BLKW 1       ;CTB LINK WORD
000002  L.NAM:'L' .BLKW 1       ;GENERIC CONTROLLER NAME (ASCII)
000004  L.DCB:'L' .BLKW 1       ;DCB ADDRESS OF THIS DEVICE
000006  L.NUM:'L' .BLKB 1       ;NUMBER OF KRB ADDRESSES IN TABLE
000007  L.STS:'L' .BLKB 1       ;CTB STATUS BYTE
000010  L.KRB:'L' .BLKW 1       ;START OF KRB ADDRESSES

;
; NOTE: THE SYMBOL $XXCTB:: IS DEFINED  FOR EACH CTB, WHERE THE
; SYMBOL IS NOT THE START OF THE CTB, BUT INSTEAD THE START OF
; THE KRB TABLE AT THE END OF THE CTB (L.KRB). THE SYMBOL XXCTB (NO"$")
; IS GENERATED BY THE DDT$ MACRO AND IS USED TO IDENTIFY THE
; WORD IN THE DRIVER WHICH CONTAINS THE ADDRESS OF THE CONTROLLER'S
; CTB IN PRIMARY POOL. XXCTB IS REFERENCE BY THE CODE GENERATED IN THE
; INTSV$ MACRO WHEN DETERMINING THE UCB ADDRESS.
;



;+
; CONTROLLER TABLE STATUS BYTE BIT DEFINITIONS
;-

LS.CLK='B'1             ;CLOCK BLOCK AT TOP OF CTB (1=YES)
LS.MDC='B'2             ;MULTIDRIVER CTB (1=YES)
LS.CBL='B'4             ;CLOCK BLK LINKED INTO CLK Q (1=YES)
```

CTBDF$

```
LS.CIN=’B’10               ;CONT. USE COMMON INT TABLE (1=YES)
LS.NET’B’=20               ;THIS IS DECNET DEVICE
                           ;ICB LISTHEAD IN K.PRM, L.DCB INVALID (1=YES)
```

A.7 DCBDF$

```
;+
;
; DEVICE CONTROL BLOCK
;
; THE DEVICE CONTROL BLOCK (DCB) DEFINES GENERIC INFORMATION
; ABOUT THE LOGICAL ACCESS TO THE DEVICE. THIS INCLUDES THE
; LOWEST AND HIGHEST LOGICAL UNIT NUMBERS (NOT THE PHYSICAL
; UNIT NUMBERS FOUND IN U.UNIT), THE LENGTH OF THE ASSOCIATED
; UCB(S), A POINTER TO THE FIRST UCB (ADDITIONAL UCBS IMPLIED
; BY D.UNIT ARE ALLOCATED CONTIGUOUSLY TO THE FIRST UCB), THE
; CLASSIFICATION OF EACH POSSIBLE I/O FUNCTION CODE, AND A
; POINTER TO THE DEVICE DRIVER AND ITS DISPATCH TABLE. THE IS
; AT LEAST ONE DCB FOR EVERY LOGICAL DEVICE NAME IN THE SYSTEM.
;
; FOR EXAMPLE THE LOGICAL DEVICE NAME 'TT' IS ASSOCIATED WITH
; THE BITMAP VIDEO (TT1:) AND THE PRINTER PORT (TT2:). BOTH ARE
; MANAGED BY THE FULL DUPLEX TERMINAL DRIVER AND SINCE
; IO.RSD/IO.WSD ARE TT1: SPECIFIC, TWO DCBS ARE USED RATHER THAN
; ONE SINCE THE FUNCTION CODE MASKS ARE DIFFERENT. IT IS
; CONCEPTUALLY POSSIBLE TO ADD A USER WRITTEN DRIVER THAT HAS THE
; LOGICAL DEVICE NAME 'TT' THOUGH IT MUST HAVE DIFFERENT LOGICAL
; UNIT NUMBERS (IE:TT3:)
;
;-

000022  .ASECT
000000  .=0
000000  D.LNK:  .BLKW   1       ; LINK TO NEXT DCB
000002  D.UCB:  .BLKW   1       ; POINTER TO FIRST UNIT CONTROL BLOCK
000004  D.NAM:  .BLKW   1       ; GENERIC DEVICE NAME
000006  D.UNIT: .BLKB   1       ; LOWEST UNIT NUMBER COVERED BY THIS
                                ; DCB
000007          .BLKB   1       ; HIGHEST UNIT NUMBER COVERED BY THIS
                                ; DCB
000010  D.UCBL: .BLKW   1       ; LENGTH OF EACH UNIT CONTROL BLOCK
                                ; IN BYTES
000012  D.DSP:  .BLKW   1       ; POINTER TO DRIVER DISPATCH TABLE
000014  D.MSK:  .BLKW   1       ; LEGAL FUNCTION MASK  CODES 0-15.
000016          .BLKW   1       ; CONTROL FUNCTION MASK CODES 0-15.
000020          .BLKW   1       ; NOP'ED FUNCTION MASK CODES 0-15.
000022          .BLKW   1       ; ACP FUNCTION MASK CODES 0-15.
000024          .BLKW   1       ; LEGAL FUNCTION MASK CODES 16.-31.
000026          .BLKW   1       ; CONTROL FUNCTION MASK CODES 16.-31.
000030          .BLKW   1       ; NOP'ED FUNCTION MASK CODES 16.-31.
000032          .BLKW   1       ; ACP FUNCTION MASK CODES 16.-31.
000034  D.PCB:  .BLKW   1       ; LOADABLE DRIVER PCB ADDRESS

                .PSECT
```

DCBDF$

```
;+
; DRIVER DISPATCH TABLE OFFSET DEFINITIONS
;-

177770   D.VTOU=-10       ; ADDRESS OF ROUTINE IN TTDRV CALLED
                          ; FOR OUTPUT COMPLETION
177772   D.VTIN=-6        ; ADDRESS OF ROUTINE IN TTDRV CALLED
                          ; FOR INPUT FROM THE CT FIRMWARE TASK
177774   D.VCHK=-4        ; ADDRESS OF ROUTINE CALLED TO VALIDATE
                          ; AND CONVERT THE LBN.  USED BY DRIVERS
                          ; THAT SUPPORT SEEK OPTIMIZATION.
177774   D.VNXC=-4        ; ADDRESS OF ROUTINE IN TTDRV CALLED TO
                          ; HAVE IT SEND THE NEXT COMMAND IN THE
                          ; TYPEAHEAD BUFFER TO MCR (NOT USED BY
                          ; P/OS)
177776   D.VDEB=-2        ; DEALLOCATE BUFFER(S)
000000   D.VINI=0         ; DEVICE INITIATOR
000002   D.VCAN=2         ; CANCEL CURRENT I/O FUNCTION
000004   D.VOUT=4         ; DEVICE TIMEOUT
000006   D.VPWF=6         ; POWERFAIL RECOVERY
000010   D.VKRB=10        ; CONTROLLER STATUS CHANGE ENTRY
000012   D.VUCB=12        ; UNIT STATUS CHANGE ENTRY


                  .IF NB  SYSDF

000014   D.VINT=14        ;BEGINNING OF INTERRUPT STUFF

                          .ENDC
```

A.8 DDT$

```
;+
; GENERATE THE I/O DEVICE DRIVER DISPATCH TABLE -- DDT$
;-

         .MACRO   DDT$     DEV,NCTRLR,INY,INX,UCBSV,NEW,BUF,OPT
         .IF NB   <OPT>
         .WORD    'DEV'CHK
         .ENDC
         .IF NB   <BUF>
         .WORD    'DEV'DEA
         .IFF
         .IF NB   <OPT>
         .WORD    172361   ;ENTRY SHOULD NOT BE USED - CRASH
         .ENDC
         .ENDC
         .ENABL   LSB
         .IF B    <INX>
$'DEV'TBL::.WORD DEV'INI
         .IFF
$'DEV'TBL::.WORD DEV'INX
         .ENDC
         .WORD    DEV'CAN
         .WORD    DEV'OUT
         .IF B    <NEW>
         .WORD    65533$
         .WORD    0
         .WORD    65531$
         .IFF
         .WORD    DEV'PWF
         .WORD    DEV'KRB
         .WORD    DEV'UCB
         .ENDC
         .IF DIF  <INY>,<NONE>
         .ASCII   /DEV/
         .IF B    <INY>
         .WORD    $'DEV'INT
         .IFF
         .IRP     X,<INY>
         .WORD    $'DEV''X
         .ENDM
         .ENDC
         .WORD    0
'DEV'CTB:         .WORD    0
         .ENDC
$'DEV'TBE::.WORD 0
         .IF NB   <UCBSV>
UCBSV:   .BLKW    NCTRLR
         .ENDC
         .IF B    <NEW>
```

```
65531$: BITB    #UC.PWF,U.CTL(R5)
        BEQ     65532$
65533$: BCS     65532$
        JMP     DEV'PWF
65532$: RETURN
        .ENDC
        .DSABL  LSB
        .ENDM
```

A.9 F11DF

```
        ;
        ; VOLUME CONTROL BLOCK
        ;

000174          .ASECT
000000  .=0

000000  V.TRCT:.BLKW    1       ; TRANSACTION COUNT
000002  V.TYPE:.BLKB    1       ; VOLUME TYPE DESCRIPTOR
000000  VT.FOR= 0               ; FOREIGN VOLUME STRUCTURE
000001  VT.SL1= 1               ; FILES-11 STRUCTURE LEVEL 1
000002  VT.SL2= 2               ; FILES-11 STRUCTURE LEVEL 2
000010  VT.ANS= 10              ; ANSI LABELED TAPE
000011  VT.UNL= 11              ; UNLABELED TAPE
000003  V.VCHA:.BLKB    1       ; VOLUME CHARACTERISTICS
000001  VC.SLK= 1               ; CLEAR VOLUME VALID ON DISMOUNT
000002  VC.HLK= 2               ; UNLOAD THE VOLUME ON DISMOUNT
000004  VC.DEA= 4               ; DEALLOCATE THE VOLUME ON DISMOUNT
000010  VC.PUB= 10              ; SET (CLEAR) US.PUB ON DISMOUNT
000020  VC.DUP= 20              ; DUPLICATE VOL NAME;
                                ; DON'T DELETE LOGICALS
000004  V.LABL:.BLKB    14      ; VOLUME LABEL (ASCII)
000020  V.PKSR:.BLKW    2       ; PACK SERIAL NUMBER FOR ERROR LOGGING
000024  V.SLEN:                 ; LENGTH OF SHORT VCB
000024  V.IFWI:.BLKW    1       ; INDEX FILE WINDOW
000026  V.FCB:.BLKW     2       ; FILE CONTROL BLOCK LISTHEAD
000032  V.IBLB:.BLKB    1       ; INDEX BIT MAP 1ST LBN HIGH BYTE
000033  V.IBSZ:.BLKB    1       ; INDEX BIT MAP SIZE IN BLOCKS
000034          .BLKW   1       ; INDEX BITMAP 1ST LBN LOW BITS
000036  V.FMAX:.BLKW    1       ; MAX NO. OF FILES ON VOLUME
000040  V.WISZ:.BLKB    1       ; DEFAULT SIZE OF WINDOW IN RTRV PTRS
                                ; VALUE IS < 128.
000041  V.SBCL:.BLKB    1       ; STORAGE BIT MAP CLUSTER FACTOR
000042  V.SBSZ:.BLKW    1       ; STORAGE BIT MAP SIZE IN BLOCKS
000044  V.SBLB:.BLKB    1       ; STORAGE BIT MAP 1ST LBN HIGH BYTE
000045  V.FIEX:.BLKB    1       ; DEFAULT FILE EXTEND SIZE
000046          .BLKW   1       ; STORAGE BIT MAP 1ST LBN LOW BITS
000050  V.VOWN:.BLKW    1       ; VOLUME OWNER'S UIC
000052  V.VPRO:.BLKW    1       ; VOLUME PROTECTION
000054  V.FPRO:.BLKW    1       ; VOLUME DEFAULT FILE PROTECTION
000056  V.FRBK:.BLKB    1       ; FREE BLOCKS ON VOLUME HIGH BYTE
000057  V.LRUC:.BLKB    1       ; AVAILABLE LRU SLOTS IN FCB LIST
000060          .BLKW   1       ; FREE BLOCKS ON VOLUME LOW BITS
000062  V.STS:.BLKB     1       ; VOLUME STATUS BYTE, CONTAINING:
000001  VS.IFW= 1               ; INDEX FILE IS WRITE ACCESSED
000002  VS.BMW= 2               ; STORAGE BITMAP FILE IS WRITE ACCESSED
000063  V.FFNU:.BLKB    1       ; FIRST FREE INDEX FILE BITMAP BLOCK
000064  V.EXT:.BLKW     1       ; POINTER TO VCB EXTENSION
000066  V.HBLB:.BLKW    2       ; LBN OF HOME BLOCK
```

```
        V.HBCS:.BLKW    2       ; HOME BLOCK CHECKSUMS
000076  V.LGTH: ; SIZE IN BYTES OF VCB

;
; MOUNT LIST ENTRY
;
; EACH ENTRY ALLOWS ACCESS TO A SPECIFIED USER FOR A NON-PUBLIC DEVICE
;
; TO ALLOW EXPANSION, ONLY THE ONLY TYPE CODE DEFINED IS "1" FOR
; DEVICE ACCESS BLOCKS
;

000076          .ASECT
000000  .=0

000000  M.LNK:.BLKW     1       ; LINK WORD
000002  M.TYPE:.BLKB    1       ; TYPE OF ENTRY
000001  MT.MLS= 1               ; MOUNTED VOLUME USER ACCESS LIST
000003  M.ACC:.BLKB     1       ; NUMBER OF ACCESSES
000004  M.DEV:.BLKW     1       ; DEVICE UCB
000006  M.TI:.BLKW      1       ; ACCESSOR TI: UCB
000010  M.LEN:  ; LENGTH OF ENTRY


;
; FILE CONTROL BLOCK
;

000010          .ASECT
000000  .=0

000000  F.LINK:.BLKW    1       ; FCB CHAIN POINTER
000002  F.FNUM:.BLKW    1       ; FILE NUMBER
000004  F.FSEQ:.BLKW    1       ; FILE SEQUENCE NUMBER
000006  F.DREF:.BLKW    1       ; DIRECTORY EOF BLOCK NUMBER
000010  F.FOWN:.BLKW    1       ; FILE OWNER'S UIC
000012  F.FPRO:.BLKW    1       ; FILE PROTECTION CODE
000014  F.UCHA:.BLKB    1       ; USER CONTROLLED CHARACTERISTICS
000015  F.SCHA:.BLKB    1       ; SYSTEM CONTROLLED CHARACTERISTICS
000016  F.HDLB:.BLKW    2       ; FILE HEADER LOGICAL BLOCK NUMBER
                                ; BEGINNING OF STATISTICS BLOCK
000022  F.LBN:.BLKW     2       ; LBN OF VIRTUAL BLOCK 1 IF CONTIGUOUS
                                ; 0 IF NON CONTIGUOUS
000026  F.SIZE:.BLKW    2       ; SIZE OF FILE IN BLOCKS
000032  F.NACS:.BLKB    1       ; NO. OF ACCESSES
000033  F.NLCK:.BLKB    1       ; NO. OF LOCKS
000012  S.STBK=.-F.LBN          ; SIZE OF STATISTICS BLOCK
000034  F.STAT:                 ; FCB STATUS WORD
000034  F.NWAC:.BLKB    1       ; NUMBER OF WRITE ACCESSORS
000035          .BLKB   1       ; STATUS BITS FOR FCB CONSISTING OF
100000  FC.WAC= 100000          ; SET IF FILE ACCESSED FOR WRITE
040000  FC.DIR= 40000           ; SET IF FCB IS IN DIRECTORY LRU
```

```
020000   FC.CEF= 20000           ; SET IF DIRECTORY EOF NEEDS UPDATING
010000   FC.FCO= 10000           ; SET IF TRYING TO FORCE DIRECTORY CONTIG
004000   FC.DNV= 4000            ; SET IF DIRECTORY NAME IS VALID
;
; IF FC.DNV IS SET, THEN THIS FCB IS ENTERED IN THE DIRECTORY FCB LRU
; CACHE. IN THIS CASE, F.DRNM IS THE FIRST WORD OF THE DIRECTORY NAME,
; F.LKL AND F.WIN ARE REUSED FOR THE SECOND AND THIRD WORDS OF THE
; DIRECTORY NAME, RESPECTIVELY, AND F.DID IS THE FILE ID OF THE
; DIRECTORY THAT THIS DIRECTORY IS ENTERED IN. IF FC.DNV IS CLEAR,
; F.DRNM IS ZERO, F.LKL IS USED AS ADVERTISED BELOW, F.WIN IS RESERVED
; FOR FUTURE USE AS THE WINDOW BLOCK LISTHEAD, AND F.DID IS ZERO.
;
000036   F.DRNM:.BLKW    1       ; 1ST WORD OF DIRECTORY NAME
000040   F.FEXT:.BLKW    1       ; POINTER TO EXTENSION FCB
000042   F.FVBN:.BLKB    1       ; HIGH ORDER BYTE: STARTING VBN
                                 ; FILE SEGMENT
000043   F.FSQN:.BLKB    1       ; FILE SEGMENT NUMBER
000044           .BLKW   1       ; LOW ORDER VBN WORD OF FILE SEGMENT
000046   F.LKL:.BLKW     1       ; POINTER TO LOCKED BLOCK LIST FOR FILE
000050   F.WIN:.BLKW     1       ; WINDOW BLOCK LIST FOR THIS FILE
000052   F.DID:.BLKW     1       ; ID OF DIRECTORY THIS DIRECTORY IS
                                 ; ENTERED IN
000054   F.LGTH:                 ; SIZE IN BYTES OF FCB


;
; WINDOW
;

000054           .ASECT
000000   .=0

000000   W.ACT:                  ; NUMBER OF ACTIVE MAPPING POINTERS
                                 ;  WHEN NO SECONDARY POOL
000000   W.BLKS:                 ; BLOCK SIZE OF SECONDARY POOL SEGMENT
                                 ;  WHEN SECONDARY POOL
000000   W.CTL:.BLKW     1       ; LOW BYTE = NO. OF MAP ENTRIES ACTIVE
                                 ; HIGH BYTE CONSISTS OF CONTROL BITS
000400   WI.RDV= 400             ; READ VIRTUAL BLOCK ALLOWED IF SET
001000   WI.WRV= 1000            ; WRITE VIRTUAL BLOCK ALLOWED IF SET
002000   WI.EXT= 2000            ; EXTEND ALLOWED IF SET
004000   WI.LCK= 4000            ; SET IF LOCKED AGAINST SHARED ACCESS
010000   WI.DLK= 10000           ; SET IF DEACCESS LOCK ENABLED
020000   WI.PND= 20000           ; WINDOW TURN PENDING BIT
040000   WI.EXL= 40000           ; SET IF MANUAL UNLOCK DESIRED
100000   WI.WCK= 100000          ; Data check all writes to file
000002   W.IOC:.BLKB     1       ; COUNT OF I/O THROUGH THIS WINDOW
000003           .BLKB   1       ; Reserved
000004   W.FCB:.BLKW     1       ; FILE CONTROL BLOCK ADDRESS
000006   W.TCB:.BLKW     1       ; TCB address of accessor
000010   W.UCB:.BLKW     1       ; Original UCB address of device
000012   W.LKL:.BLKW     1       ; POINTER TO LIST OF USERS LOCKED BLOCKS
```

```
000014   W.WIN:.BLKW     1      ; WINDOW BLOCK LIST LINK WORD

         .IF     NB,SYSDEF      ; IF SYSDEF SPECIFIED IN CALL
         .IF     NDF,P$$WND     ; IF SECONDARY POOL WINDOWS NOT ALLOWED

;
; NON-SECONDARY POOL WINDOW BLOCK
;        IF SECONDARY POOL WINDOWS ARE NOT ENABLED, THE WINDOW BLOCK
;        CONTAINS THE CONTROL INFORMATION AND RETRIEVAL POINTERS.
;

W.VBN:   .BLKB   1       ; HIGH BYTE OF 1ST VBN MAPPED BY WINDOW
W.MAP:                   ; DEFINE LABEL WITH ODD ADDRESS TO CATCH
                        ; BAD REFS
W.WISZ:  .BLKB   1       ; SIZE IN RTRV PTRS OF WINDOW (7 BITS)
         .BLKW   1       ; LOW ORDER WORD OF 1ST VBN MAPPED
W.RTRV:                  ; OFFSET TO 1ST RETRIEVAL POINTER
                        ; IN WINDOW

W.SLEN=-4        ; Dummy definition to prevent incorrect reference
                 ; (-4 when rounded "up" is a VERY large block)

.IFF     ; IF WINDOWS IN SECONDARY POOL

;
; SECONDARY POOL WINDOW CONTROL AND MAPPING BLOCK
;        IF SECONDARY POOL WINDOW BLOCKS ARE ENABLED, LUTN2 POINTS
;        TO A CONTROL BLOCK IN SYSTEM POOL WHICH CONTAINS THE
;        FOLLOWING CONTROL FIELDS AND THE MAPPING INFORMATION
;        FOR THE SECONDARY POOL WINDOW.
;

000016   W.MAP:.BLKW     1      ; ADDR TO THE MAPPING
                                ; PTRS IN SECONDARY POOL
000020   W.SLEN:                ; Length of primary pool stub

;
; SECONDARY POOL WINDOW
;        IF SECONDARY POOL WINDOW BLOCKS ARE ENABLED, THE RETRIEVAL
;        POINTERS ARE MAINTAINED IN SECONDARY POOL IN THE FOLLOWING
;        FORMAT.
;

000000   .=0

000000           ASSUME  W.CTL,0
         .IIF NE <W.CTL>-<0>    .ERROR  ;EXPRESSIONS NOT EQUAL

000000           .BLKB   1      ; NUMBER OF ACTIVE MAPPING POINTERS
000001   W.USE:.BLKB     1      ; STATUS OF BLOCK
000002   W.VBN:.BLKB     1      ; HIGH BYTE OF 1ST VBN MAPPED BY WINDOW
```

```
000003   W.WISZ:.BLKB    1       ; SIZE IN RTRV PTRS OF WINDOW (7 BITS)
000004           .BLKW   1       ; LOW ORDER WORD OF 1ST VBN MAPPED
000006   W.RTRV:                 ; OFFSET TO 1ST RETRIEVAL POINTER IN WIND

         .ENDC    ; END SECONDARY POOL WINDOW CONDITIONAL

         .ENDC    ; END SYSDEF CONDITIONAL

;
; LOCKED BLOCK LIST NODE
;

000006           .ASECT
000000   .=0

000000   L.LNK:.BLKW     1       ; LINK TO NEXT NODE IN LIST
000002   L.WI1:.BLKW     1       ; POINTER TO WINDOW FOR FIRST ENTRY
000004   L.VB1:.BLKB     1       ; HIGH ORDER VBN BYTE
000005   L.CNT:.BLKB     1       ; COUNT FOR ENTRY
000006           .BLKW   1       ; LOW ORDER VBN
000010   L.LKSZ:

;
; END OF DEFINITIONS
;
```

A.10 GTPKT$

```
;
; GET I/O PACKET MACRO -- AUTOMATES UNIT DETERMINATION -- GTPKT$
;

        .MACRO  GTPKT$  DEV,NCTRLR,ADDR,UCBSV,SUC
        CALL    $GTPKT
        .IF B   <ADDR>
        BCC     65535$
        RETURN
65535$:
        .IFF
        BCS     ADDR
        .ENDC
        .IF B   <UCBSV>
        .IF B   <SUC>
        MOV     R5,S.OWN(R4)
        .ENDC
        .IFF
        .IF GT  NCTRLR-1
        MOV     R5,UCBSV(R3)
        .IFF
        MOV     R5,UCBSV
        .ENDC
        .ENDC
        .ENDM
```

A.11 HDRDF$

```
;+
; TASK HEADER OFFSET DEFINITIONS
;-

                      .ASECT
         .=0
000000   H.CSP: .BLKW    1       ;CURRENT STACK POINTER
000002   H.HDLN:.BLKW    1       ;HEADER LENGTH IN BYTES
000004   H.SMAP:.BLKB    1       ;SUPERVISOR D SPACE OVERMAP MASK
000005   H.DMAP:.BLKB    1       ;USER D SPACE OVERMAP MASK
000006          .BLKW    1       ;RESERVED
000010   H.CUIC:.BLKW    1       ;CURRENT TASK UIC
000012   H.DUIC:.BLKW    1       ;DEFAULT TASK UIC
000014   H.IPS: .BLKW    1       ;INITIAL PROCESSOR STATUS WORD (PS)
000016   H.IPC: .BLKW    1       ;INITIAL PROGRAM COUNTER (PC)
000020   H.ISP: .BLKW    1       ;INITIAL STACK POINTER (SP)
000022   H.ODVA:.BLKW    1       ;ODT SST VECTOR ADDRESS
000024   H.ODVL:.BLKW    1       ;ODT SST VECTOR LENGTH
000026   H.TKVA:.BLKW    1       ;TASK SST VECTOR ADDRESS
000030   H.TKVL:.BLKW    1       ;TASK SST VECTOR LENGTH
000032   H.PFVA:.BLKW    1       ;POWER FAIL AST CONTROL BLOCK ADDRESS
000034   H.FPVA:.BLKW    1       ;FLOATING POINT AST CONTROL BLOCK ADDR
000036   H.RCVA:.BLKW    1       ;RECEIVE AST CONTROL BLOCK ADDRESS
000040   H.EFSV:.BLKW    1       ;EVENT FLAG ADDRESS SAVE ADDRESS
000042   H.FPSA:.BLKW    1       ;POINTR TO FLOATING POINT/EAE SAVE AREA
000044   H.WND: .BLKW    1       ;POINTER TO NUMBER OF WINDOW BLOCKS
000046   H.DSW: .BLKW    1       ;TASK DIRECTIVE STATUS WORD
000050   H.FCS: .BLKW    1       ;FCS IMPURE POINTER
000052   H.FORT:.BLKW    1       ;FORTRAN IMPURE POINTER
000054   H.OVLY:.BLKW    1       ;OVERLAY IMPURE POINTER
000056   H.VEXT:.BLKW    1       ;WORK AREA EXTENSION VECTOR POINTER
000060   H.SPRI:.BLKB    1       ;PRIORITY DIFFERENCE FOR SWAPPING
000061   H.NML: .BLKB    1       ;NETWORK MAILBOX LUN
000062   H.RRVA:.BLKW    1       ;RECEIVE BY REF AST CONTROL BLOCK ADDR
000064   H.X25: .BLKB    1       ;FOR USE BY X25 SOFTWARE
000065          .BLKB    1       ;5 RESERVED BYTES
000066          .BLKW    2       ;
000072   H.GARD:.BLKW    1       ;POINTER TO HEADER GUARD WORD
000074   H.NLUN:.BLKW    1       ;NUMBER OF LUN'S
000076   H.LUN: .BLKW    2       ;START OF LOGICAL UNIT TABLE

;+
; LENGTH OF FLOATING POINT SAVE AREA
;-

         H.FPSL=25.*2

;+
; WINDOW BLOCK OFFSETS
```

HDRDF$

```
;-

        .=0
000000  W.BPCB:.BLKW   1       ;PARTITION CONTROL BLOCK ADDRESS
000002  W.BLVR:.BLKW   1       ;LOW VIRTUAL ADDRESS LIMIT
000004  W.BHVR:.BLKW   1       ;HIGH VIRTUAL ADDRESS LIMIT
000006  W.BATT:.BLKW   1       ;ADDRESS OF ATTACHMENT DESCRIPTOR
000010  W.BSIZ:.BLKW   1       ;SIZE OF WINDOW IN 32W BLOCKS
000012  W.BOFF:.BLKW   1       ;PHYSICAL MEMORY OFFSET IN 32W BLOCKS
000014  W.BFPD:.BLKB   1       ;FIRST PDR ADDRESS
000015  W.BNPD:.BLKB   1       ;NUMBER OF PDRs TO MAP
000016  W.BLPD:.BLKW   1       ;CONTENTS OF LAST PDR
000020  W.BLGH:                ;LENGTH OF WINDOW DESCRIPTOR

;
; BIT DEFINITION FOR W.BLPD
;

        WB.NBP=20              ;CACHE BYPASS IS NOT DESIRED FOR THIS
                               ;WINDOW
        WB.BPS=40              ;ALWAYS BYPASS THE CACHE FOR THIS
                               ;WINDOW

                .PSECT
```

A.12 HWDDF$

```
;+
; MACROS FOR DEFINING MAPPING REGISTER DEFINITIONS
;-

        .MACRO  CRESET  NAM,ADDR
        $$$=0
        .REPT   8.
        CRENAM  NAM,ADDR+<$$$*2>,\$$$
        $$$=$$$+1
        .ENDR
        .ENDM

        .MACRO  CRENAM  NAM,ADDR,N
        'NAM''N'==ADDR
        .ENDM

;+
; HARDWARE REGISTER ADDRESSES AND STATUS CODES
;-

177746  MPCSR=177746    ;ADDRESS OF PDP-11/70 MEMORY PARITY REGISTER
172100  MPAR=172100     ;ADDRESS OF FIRST MEMORY PARITY REGISTER
177772  PIRQ=177772     ;PROGRAMMED INTERRUPT REQUEST REGISTER
000000  PR0=0           ;PROCESSOR PRIORITY 0
000040  PR1=40          ;PROCESSOR PRIORITY 1
000200  PR4=200         ;PROCESSOR PRIORITY 4
000240  PR5=240         ;PROCESSOR PRIORITY 5
000300  PR6=300         ;PROCESSOR PRIORITY 6
000340  PR7=340         ;PROCESSOR PRIORITY 7
177776  PS=177776       ;PROCESSOR STATUS WORD
177570  SWR=177570      ;CONSOLE SWITCH AND DISPLAY REGISTER
177564  TPS=177564      ;CONSOLE TERMINAL PRINTER STATUS REGISTER

;+
; EXTENDED ARITHMETIC ELEMENT REGISTERS
;-

        .IF DF  E$$EAE

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

        .ENDC


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

HWDDF$

172340	KINAR0
172342	KINAR1
172344	KINAR2
172346	KINAR3
172350	KINAR4
172352	KINAR5
172354	KINAR6
172356	KINAR7
172300	KINDR0
172302	KINDR1
172304	KINDR2
172306	KINDR3
172310	KINDR4
172312	KINDR5
172314	KINDR6
172316	KINDR7
172360	KDSAR0
172362	KDSAR1
172364	KDSAR2
172366	KDSAR3
172370	KDSAR4
172372	KDSAR5
172374	KDSAR6
172376	KDSAR7
172320	KDSDR0
172322	KDSDR1
172324	KDSDR2
172326	KDSDR3
172330	KDSDR4
172332	KDSDR5
172334	KDSDR6
172336	KDSDR7
172240	SISAR0
172242	SISAR1
172244	SISAR2
172246	SISAR3
172250	SISAR4
172252	SISAR5
172254	SISAR6
172256	SISAR7
172200	SISDR0
172202	SISDR1
172204	SISDR2
172206	SISDR3
172210	SISDR4
172212	SISDR5
172214	SISDR6
172216	SISDR7
172260	SDSAR0
172262	SDSAR1
172264	SDSAR2

HWDDF$

172266	SDSAR3
172270	SDSAR4
172272	SDSAR5
172274	SDSAR6
172276	SDSAR7
172220	SDSDR0
172222	SDSDR1
172224	SDSDR2
172226	SDSDR3
172230	SDSDR4
172232	SDSDR5
172234	SDSDR6
172236	SDSDR7
177640	UINAR0
177642	UINAR1
177644	UINAR2
177646	UINAR3
177650	UINAR4
177652	UINAR5
177654	UINAR6
177656	UINAR7
177600	UINDR0
177602	UINDR1
177604	UINDR2
177606	UINDR3
177610	UINDR4
177612	UINDR5
177614	UINDR6
177616	UINDR7
177660	UDSAR0
177662	UDSAR1
177664	UDSAR2
177666	UDSAR3
177670	UDSAR4
177672	UDSAR5
177674	UDSAR6
177676	UDSAR7
177620	UDSDR0
177622	UDSDR1
177624	UDSDR2
177626	UDSDR3
177630	UDSDR4
177632	UDSDR5
177634	UDSDR6
177636	UDSDR7
172340	KISAR0
172342	KISAR1
172344	KISAR2
172346	KISAR3
172350	KISAR4
172352	KISAR5

```
172354   KISAR6
172356   KISAR7
172300   KISDR0
172302   KISDR1
172304   KISDR2
172306   KISDR3
172310   KISDR4
172312   KISDR5
172314   KISDR6
172316   KISDR7
177640   UISAR0
177642   UISAR1
177644   UISAR2
177646   UISAR3
177650   UISAR4
177652   UISAR5
177654   UISAR6
177656   UISAR7
177600   UISDR0
177602   UISDR1
177604   UISDR2
177606   UISDR3
177610   UISDR4
177612   UISDR5
177614   UISDR6
177616   UISDR7


170200   UBMPR=170200     ;UNIBUS MAPPING REGISTER 0
140000   CMODE=140000     ;CURRENT MODE FIELD OF PS WORD
030000   PMODE=30000      ;PREVIOUS MODE FIELD OF PS WORD
040000   CSMODE=40000     ;CURRENT MODE = SUPERVISOR PS WORD BITS
010000   PSMODE=10000     ;PREVIOUS MODE = SUPERVISOR PS WORD BITS
177572   SR0=177572       ;SEGMENT STATUS REGISTER 0
172516   SR3=172516       ;SEGMENT STATUS REGISTER 3
177766   CPUERR=177766    ;CPU ERROR REGISTER
177744   MEMERR=177744    ;MEMORY SYSTEM ERROR REGISTER
177746   MEMCTL=177746    ;MEMORY CONTROL REGISTER

;+
; DEFINE THE LOCATIONS USED IN THE NON-VOLATILE RAM (NVR)
; FOR XT SYSTEMS
;-

173054   N.KEY=173054     ;NUMBER OF KEYS PRESSED
173064   N.UPT=173064     ;UPTIME IN MINUTES
173074   N.DZA=173074     ;NUMBER OF I/OS DONE ON THE DZ
173104   N.DWA=173104     ;NUMBER OF I/OS DONE ON THE DW
173114   N.DAY=173114     ;DATE THAT THE NVR WAS LAST INITIALIZED
173116   N.MON=173116     ;...
173120   N.YEA=173120     ;...
```

HWDDF$

```
;+
; FEATURE SYMBOL DEFINITIONS
;-

000001   FE.EXT=1          ;22-BIT EXTENDED MEMORY SUPPORT
000002   FE.MUP=2          ;MULTI-USER PROTECTION SUPPORT
000004   FE.EXV=4          ;EXECUTIVE IS SUPPORTED TO 20K
000010   FE.DRV=10         ;LOADABLE DRIVER SUPPORT
000020   FE.PLA=20         ;PLAS SUPPORT
000040   FE.CAL=40         ;DYNAMIC CHECKPOINT SPACE ALLOCATION
000100   FE.PKT=100        ;PREALLOCATION OF I/O PACKETS
000200   FE.EXP=200        ;EXTEND TASK DIRECTIVE SUPPORTED
000400   FE.LSI=400        ;PROCESSOR IS AN LSI-11
001000   FE.OFF=1000       ;PARENT/OFFSPRING TASKING SUPPORTED
002000   FE.FDT=2000       ;FULL DUPLEX TERMINAL DRIVER SUPPORTED
004000   FE.X25=4000       ;X.25 CEX IS LOADED
010000   FE.DYM=10000      ;DYNAMIC MEMORY ALLOCATION SUPPORTED
020000   FE.CEX=20000      ;COM EXEC IS LOADED
040000   FE.MXT=40000      ;MCR EXIT AFTER EACH COMMAND MODE
100000   FE.NLG=100000     ;LOGINS DISABLED - MULTI-USER SUPPORT

;+
; FEATURE MASK DEFINITIONS (SECOND WORD)
;-

000001   F2.DAS=1          ;KERNEL DATA SPACE SUPPORTED
000002   F2.LIB=2          ;SUPERVISOR MODE LIBRARIES SUPPORTED
000004   F2.MP=4           ;SYSTEM SUPPORTS MULTIPROCESSING
000010   F2.EVT=10         ;SYSTEM SUPPORTS EVENT TRACE FEATURE
000020   F2.ACN=20         ;SYSTEM SUPPORTS CPU ACCOUNTING
000040   F2.SDW=40         ;SYSTEM SUPPORTS SHADOW RECORDING
000100   F2.POL=100        ;SYSTEM SUPPORTS SECONDARY POOLS
000200   F2.WND=200        ;SYSTEM SUPPORTS SECONDARY POOL FILE WINDOWS
000400   F2.DPR=400        ;SYSTEM HAS A SEPARATE DIRECTIVE PARTITION
001000   F2.IRR=1000       ;INSTALL, RUN, AND REMOVE SUPPORT
002000   F2.GGF=2000       ;GROUP GLOBAL EVENT FLAG SUPPORT
004000   F2.RAS=4000       ;RECEIVE/SEND DATA PACKET SUPPORT
010000   F2.AHR=10000      ;ALT. HEADER REFRESH AREA SUPPORT
020000   F2.RBN=20000      ;ROUND ROBIN SCHEDULING SUPPORT
040000   F2.SWP=40000      ;EXECUTIVE LEVEL DISK SWAPPING SUPPORT
100000   F2.STP=100000     ;EVENT FLAG MASK IS IN THE TCB(1=YES)

;+
; THIRD FEATURE MASK SYMBOL DEFINITIONS
;-

000001   F3.CRA=1          ;SYSTEM SPONTANEOUSLY CRASHED (1=YES)
000002   F3.XCR=2          ;SYSTEM CRASHED FROM XDT (1=YES)
000004   F3.EIS=4          ;SYSTEM REQUIRES EXTENDED INSTRUCTION SET
000010   F3.STM=10         ;SYSTEM HAS SET SYSTEM TIME DIRECTIVE
000020   F3.UDS=20         ;SYSTEM SUPPORTS USER DATA SPACE
```

```
000040   F3.PRO=40        ;SYSTEM SUPPORTS SEC. POOL PROTO TCBS
000100   F3.XHR=100       ;SYSTEM SUPPORTS EXTERNAL TASK HEADERS
000200   F3.AST=200       ;SYSTEM HAS AST SUPPORT
000400   F3.11S=400       ;RSX-11S SYSTEM
001000   F3.CLI=1000      ;MULTIPLE CLI SUPPORT
002000   F3.TCM=2000      ;SYSTEM HAS SEPARATE TERMINAL DRIVER POOL
004000   F3.PMN=4000      ;SYSTEM SUPPORTS POOL MONITORING
010000   F3.WAT=10000     ;SYSTEM HAS WATCHDOG TIMER SUPPORT
020000   F3.RLK=20000     ;SYSTEM SUPPORTS RMS RECORD LOCKING
040000   F3.SHF=40000     ;SYSTEM SUPPORTS SHUFFLER TASK

;+
; FOURTH FEATURE MASK BITS
;-

000001   F4.CXD=1         ;COMM EXEC IS DEALLOCATED (NON-I/D ONLY)
000002   F4.XT=2          ;SYSTEM IS AN XT SYSTEM (1=YES)
000002   F4.POS=F4.XT     ;SYNONYM -- SYSTEM IS A P/OS SYSTEM (1=YES)
000004   F4.ERL=4         ;SYSTEM SUPPORTS ERROR LOGGING (1=YES)
000010   F4.PTY=10        ;SYSTEM SUPPORTS PARITY MEMORY (1=YES)
000020   F4.DVN=20        ;SYSTEM SUPPORTS DECIMAL VERSIONS (1=YES)
000040   F4.LCD=40        ;SYSTEM SUPPORTS LOADABLE CRASH (1=YES)
000100   F4.NIM=100       ;SYSTEM SUPPORTS DELETED TASK IMAGES (1=YES)
000200   F4.CHE=200       ;SYSTEM SUPPORTS DISK DATA CACHING (1=YES)
000400   F4.LOG=400       ;SYSTEM SUPPORTS LOGICAL NAMES (1=YES)
001000   F4.NAM=1000      ;SYSTEM SUPPORTS NAMED DIRECTORIES (1=YES)
002000   F4.FMP=2000      ;SYSTEM SUPPORTS FAST MAP DIRECTIVE
004000   F4.DCL=4000      ;DCL IS DEFAULT CLI (1=YES)
010000   F4.DDS=10000     ;NAMED DIRECTORY MODE IS THE DEFAULT (1=YES)
020000   F4.ACD=20000     ;SYSTEM SUPPORTS ACDs (1=YES)
040000   F4.NCT=40000     ;SYSTEM HAS NCT SUPPORT (1=YES)
100000   F4.LSD=100000    ;SYSTEM HAS LUT SCAN DISABLED
000001   F5.P3X=1         ;SYSTEM SUPPORTS PROFESSIONAL 3XX SERIES

;+
; HARDWARE FEATURE MASK BIT DEFINITIONS
;
;        HF.CIS,HF.FPP DEFINED AS SIGN BITS FOR RUN TIME SPEED
;-

000001   HF.UBM=1         ;PROCESSOR HAS A UNIBUS MAP (1=YES)
000002   HF.EIS=2         ;PROCESSOR HAS EXTENDED INSTRUCION SET
000004   HF.QB=4          ;SYSTEM HAS A QBUS (1=YES)
000010   HF.DSP=10        ;HARDWARE SUPPORTS DATA SPACE
000200   HF.CIS=200       ;PROCESSOR SUPPORTS COMMERCIAL INSTRUCTION SET
100000   HF.FPP=100000    ;(1=PROC. HAS NO FLOATING POINT UNIT)

;+
; SECOND HARDWARE FEATURE MASK BIT DEFINITIONS
; THIS WORD IS RESERVED FOR XT HARDWARE FEATURES
;-
```

```
000001   H2.NVR=1        ;XT NON-VOLATILE RAM PRESENT (1=YES)
000002   H2.INV=2        ;NON-VOLATILE RAM IS INVALID (1=YES)
000004   H2.CLK=4        ;XT CLOCK IS PRESENT (1=YES)
000010   H2.ITF=10       ;INVALID TIME FORMAT IN NON-VOLATILE RAM
                        ;(1=YES)
000020   H2.PRO=20       ;RUNNING ON PRO/3XX HARDWARE
000040   H2.WS=40        ;SYSTEM IS A WORKSTATION (1=YES)
000100   H2.FS=100       ;SYSTEM IS A FILESERVER (1=YES)
100000   H2.BRG=100000   ;XT BRIDGE MODULE PRESENT (1=YES)

;+
; SYSGEN FEATURE SELECTIONS MASK. THIS IS INTENDED TO RECORD IN A
; BIT MASK THE CHOICES THE USER HAS MADE AT SYSGEN TIME. FEATURES WILL
; BE LISTED HERE WHEN THEY ARE BEING RECORDED FOR OUR INFORMATIONAL
; PURPOSES ONLY. THEY CANNOT BE TESTED LIKE BITS IN THE FEATURE MASK
; SINCE THIS ONLY EXISTS IN THE RSX11M.STB FILE. NO BITS IN MEMORY ARE
; USED. THEY ARE ONLY INTENDED TO BE PRINTED FROM THE STB FILE BY CDA.
;-

000001   SF.STD=1        ;STANDARD EXEC SELECTED
000002   SF.PGN=2        ;SYSTEM WAS PRE-GENERATED (EX.
                        ;RL02/RC25 SYSTEM)

;+
; MULTIPROCESSOR STATUS TABLE DEFINITIONS (TEMPORARY)
;-

100000   MP.CRH=100000   ;CRASH PROCESSOR IMMEDIATELY
040000   MP.PWF=40000    ;POWERFAIL ON ONE CPU
020000   MP.RSM=20000    ;RESET INTERRUPT MASKS
010000   MP.NOP=10000    ;NOP FUNCTION FOR TRANSMISSION CHECK
000004   MP.STP=4        ;STOP PROCESSOR IN ORDERLY FASHION
007777   MP.INT=7777     ;BIC MASK FOR INTERRUPT LVL FUNCTIONS

         .MACRO  HWDDF$  X,Y,Z
         .ENDM
```

A.13 INTSV$

```
;
; INTERRUPT SAVE GENERATION FOR NON-ERROR LOGGING DEVICES -- INTSV$
;

        .MACRO  INTSV$  DEV,PRI,NCTRLR,PSWSV,UCBSV
        GTUCB$  UCBSV,NCTRLR,DEV
        .ENDM


;
; GENERATE CODE TO LOAD UCB ADDRESS INTO R5 -- CALLED
; ONLY BY INTSV$, AND TTSET$ (IN TTDRV).
;

        .MACRO  GTUCB$  UCBSV,NCTRLR,DEV
        .IF NB  <UCBSV>
        .IF GT  NCTRLR-1
        MOV     UCBSV(R4),R5
        .IFF
        MOV     UCBSV,R5
        .ENDC
        .IFF
        MOV     'DEV'CTB,R5     ;;;GET ADDRESS OF KRB TABLE IN CTB
        ADD     R4,R5           ;;;ADD CONTROLLER INDEX
        MOV     (R5),R5         ;;;GET KRB ADDRESS FROM CTB
        MOV     K.OWN(R5),R5    ;;;RETRIEVE OWNER'S UCB ADDRESS
        .ENDC
        .ENDM
```

A.14 ITBDF$

```
;+
; INTERRUPT TRANSFER BLOCK (ITB) OFFSET DEFINITIONS
;-

                 .IF DF   A$$TRP

                 .MCALL   PKTDF$
                 PKTDF$           ; DEFINE AST BLOCK OFFSETS

                 .ENDC

                 .ASECT
        .=0
000000  X.LNK:   .BLKW    1       ; LINK WORD FOR ITB LIST STARTNG IN
                                  ; TCB
000002  X.JSR: JSR        R5,@#0  ; CALL $INTSC
000006  X.PSW:   .BLKB    1       ; LOW BYTE OF PSW FOR ISR
000007           .BLKB    1       ; UNUSED
000010  X.ISR:   .BLKW    1       ; ISR ENTRY POINT (APR5 MAPPING)
000012  X.FORK:                   ; FORK BLOCK
000012           .BLKW    1       ; THREAD WORD
000014           .BLKW    1       ; FORK PC
000016           .BLKW    1       ; SAVED R5
000020           .BLKW    1       ; SAVED R4

                 .IF DF   M$$MGE

        X.REL: .BLKW      1       ; RELOCATION BASE FOR APR5

                 .ENDC

000022  X.DSI:   .BLKW    1       ; ADDRESS OF DIS.INT. ROUTINE
000024  X.TCB:   .BLKW    1       ; TCB ADDRESS OF OWNING TASK

                 .IF NB   SYSDF

                 .IF DF   A$$TRP

                 .BLKW    1       ; A.DQSR FOR AST BLOCK
        X.AST:   .BLKB    A.PRM   ; AST BLOCK

                 .ENDC

000026  X.VEC:   .BLKW    1       ; VECTOR ADDRESS (IF AST SUPPORT,
                                  ; THIS IS FIRST AND ONLY AST PARAMETER)
000030  X.VPC:   .BLKW    1       ; SAVED VECTOR PC
000032  X.LEN:                    ; LENGTH IN BYTES OF ITB

                 .ENDC
```

ITBDF$

```
.PSECT
```

A.15 KRBDF$

```
;+
; CONTROLLER REQUEST BLOCK (KRB)
;
; THE CONTROLLER REQUEST BLOCK DEFINES THE ENVIRONMENT OF A
; DEVICE CONTROLLER.  ONE KRB EXISTS FOR EVERY DEVICE CONTROLLER
; IN A P/OS SYSTEM.  THE KRB CONTAINS CERTAIN DEVICE STATUS
; INCLUDING THE CSR -- FIRST DEVICE REGISTER, VECTOR ADDRESS,
; INTERUPT CONTROLLER 'A' CSR AND THE SLOT NUMBER FOR THE
; CONTROLLER.
;-
         .ASECT
.=177764
177764   K.PRM:  .BLKW   1      ;DEVICE DEPENDANT PARAMETER WORD
177766   K.ICSR: .BLKW   1      ;INTERRUPT 'A' CONTROLLER CSR (ADD 6
                                ;FOR ICSR IF APPROPRIATE TO DEVICE)
177770   K.SLT:  .BLKB   1      ;SLOT NUMBER
177771           .BLKB   1      ;RESERVED
177772   K.PRI:  .BLKB   1      ;CONTROLLER PRIORITY
177773   K.VCT:  .BLKB   1      ;INTERRUPT VECTOR ADDRESS
177774   K.CON:  .BLKB   1      ;CONTROLLER INDEX WITHIN THE SYSTEM
177775   K.IOC:  .BLKB   1      ;CONTROLLER I/O COUNT
177776   K.STS:  .BLKW   1      ;CONTROLLER STATUS
000000   K.CSR:  .BLKW   1      ;ADDRESS OF CONTROL STATUS REGISTER
         ;
         ; NOTE: K.CSR MUST BE THE ZERO OFFSET!
         ;
000002   K.OFF:  .BLKW   1      ;OFFSET TO UCB/UMR/RHBAE TABLE
000004   K.HPU:  .BLKB   1      ;HIGHEST PHYSICAL UNIT NUMBER
000005           .BLKB   1      ;UNUSED BYTE
000006   K.OWN:  .BLKW   1      ;OWNER OF CONTROLLER
000010   K.CRQ:  .BLKW   2      ;CONTROLLER REQUEST QUEUE
000014   K.URM:  .BLKW   1      ;RESERVED FOR FUTURE USE
000016   K.FRK:  .BLKW   1      ;POSSIBLE KRB FORK BLOCK

;+
; OFFSETS FOR THE KRB EXTENSION REACHED BY ADDING (K.OFF) TO
; THE STARTING ADDRESS OF THE KRB.
;
;
; WHEN ONE ADDS (K.OFF) TO THE KRB ADDRESS, IT YIELDS AN
; ADDRESS WHICH POINTS TO HERE.
;
;-
000020   KE.UCB: .BLKW   1      ;OFFSET TO UCB TABLE (IF KS.UCB SET)

                 .PSECT

;+
; CONTROLLER REQUEST BLOCK (KRB) STATUS BIT DEFINITIONS
```

KRBDF$

```
;-

000001   KS.OFL=1                    ;CONTROLLER OFFLINE (1=YES)
000002   KS.MOF=2                    ;CONTROLLER MARKED FOR OFFLINE (1=YES)
000004   KS.UOP=4                    ;SUPPORTS OVERLAPPED OPERATION (1=YES)
000010   KS.MBC=10                   ;(RESERVED)
000020   KS.SDX=20                   ;SEEKS ALLOWED DURING DATA XFERS
                                     ;       (1=YES)
000040   KS.POE=40                   ;PARALLEL OPERATION ENABLED (1=YES)
000100   KS.UCB=100                  ;UCB TABLE PRESENT (1=YES)
000200   KS.DIP=200                  ;DATA TRANSFER IN PROGRESS (1=YES)
000400   KS.PDF=400                  ;PRIVILEGED DIAGNOSTIC FUNCTIONS ONLY
                                     ;       (1=YES)
001000   KS.EXT=1000                 ;EXTENDED 22-BIT UNIBUS CONTROLLER
                                     ;       (1=YES)
002000   KS.SLO=2000                 ;CONTROLLER IS SLOW COMING ONLINE
                                     ;       (1=YES)

;+
; DEFINE THE CONTIGUOUS SCB OFFSETS
;-

000022           .ASECT
         .=177756
177756   S.ICSR:.BLKW    1           ;INTERRUPT 'A' CONTROLLER CSR
177760   S.SLT: .BLKB    1           ;SLOT NUMBER
177761          .BLKB    1           ;RESERVED
177762   S.PRI: .BLKB    1           ;CONTROLLER PRIORITY
177763   S.VCT: .BLKB    1           ;INTERRUPT VECTOR ADDRESS
177764   S.CON: .BLKB    1           ;CONTROLLER INDEX
177765          .BLKB    1
177766          .BLKW    1
177770   S.CSR: .BLKW    1           ;CONTROL AND STATUS REGISTER
177772          .BLKW    1
177774          .BLKB    1
177775          .BLKB    1
177776   S.OWN: .BLKW    1           ;DISTRIBUTED CNTBL


;+
; SUBCONTROLLER REQUEST BLOCK (KRB1)
;
; THE SUBCONTROLLER REQUEST BLOCK DEFINES THE ENVIRONMENT OF
; A DEVICE SUBCONTROLLER.  EXACTLY ONE KRB1 EXISTS FOR EVERY
; DEVICE SUBCONTROLLER IN AN RSX-11M+ SYSTEM.
;-
                 .ASECT

         .=-4
177774   K1.CON:.BLKB    1           ;SUBCONTROLLER INDEX WITHIN THE SYSTEM
```

KRBDF$

```
177775          .BLKB     1       ;UNUSED BYTE
177776  K1.STS:.BLKW      1       ;SUBCONTROLLER STATUS
000000  K1.MAS:.BLKW      1       ;UCB ADDRESS OF THE MASTER UNIT
        ;
        ;. NOTE: K1.MAS MUST BE THE ZERO OFFSET
        ;
000002  K1.OWN:.BLKW      1       ;OWNER OF SUBCONTROLLER
000004  K1.CRQ:.BLKW      2       ;SUBCONTROLLER REQUEST QUEUE
000010  K1.UCB:                   ;START OF THE UCB TABLE (IF ANY)

                .PSECT
```

A.16 LBLDF$

```
;+
;
; TASK IMAGE FILE LABEL BLOCK DEFINITIONS
;
; RESIDENT LIBRARY DESCRIPTOR OFFSETS
;
;-

000010          .ASECT

         000000  .=0
000000   R$LNAM:.BLKW    2      ; RADIX-50 LIBRARY NAME
000004   R$LSA: .BLKW    1      ; LIBRARY STARTING VIRTUAL ADDRESS
000006   R$LHGV:.BLKW    1      ; LIBRARY ADDRESS WINDOW 0 BOUND
000010   R$LMXV:.BLKW    1      ; LIBRARY HIGH VIRTUAL ADDRESS LIMIT
000012   R$LLDZ:.BLKW    1      ; LIBRARY LOAD SIZE (32W BLOCKS)
000014   R$LMXZ:.BLKW    1      ; LIBRARY MAX. SIZE (32W BLOCKS)
000016   R$LOFF:.BLKW    1      ; LIBRARY OFFSET INTO PARTITION
                                ; (32W BLOCKS)
000020   R$LWND:.BLKW    1      ; NUMBER OF LIBRARY ADDRESS WINDOWS
000022   R$LSEG:.BLKW    1      ; SIZE OF LIBRARY SEGMENT DESCRIPTORS
000024   R$LFLG:.BLKW    1      ; LIBRARY FLAGS WORD
000026   R$LDAT:.BLKW    3      ; LIBRARY CREATION DATE (YR., MO., DAY)
000034   R$LSIZ:.BLKW    0      ; LENGTH OF LIBRARY DESCRIPTOR
         ;
         ; LIBRARY LIST ENTRY FLAGS
         ;
100000   LD$ACC=100000   ; ACCESS INTENT (1=RW, 0=RO)
040000   LD$RSV=040000   ; APR RESERVATION FLAG (1=APR RESERVED)
020000   LD$CLS=020000   ; LIBRARY IS PART OF A CLUSTER
000010   LD$SUP=000010   ; SUPERVISOR MODE LIBRARY (1=YES)
000004   LD$REL=000004   ; PIC FLAG (1=POSITION INDEPENDANT)
000002   LD$TYP=000002   ; SHARED REGION IS LIB (=0) OR COMMON (=1)

         ;
         ; LABEL BLOCK OFFSETS
         ;

000000   .=0
000000   L$BTSK:.BLKW    2      ; RADIX-50 TASK NAME

;
;        *** NOTE ***
;
; LABEL BLOCK PARAMETERS BETWEEN THIS OFFSET AND THE START OF THE
; TASK LIBRARY DESCRIPTORS MUST BE IDENTICAL IN FORMAT AND CONTENT
; TO A RESIDENT LIBRARY DESCRIPTOR ENTRY.
;
```

LBLDF$

```
000004   L$BPAR:.BLKW    2       ; RADIX-50 PARTITION NAME
000010   L$BSA: .BLKW    1       ; TASK STARTING VIRTUAL ADDRESS
000012   L$BHGV:.BLKW    1       ; WINDOW 0 VIRTUAL ADDRESS LIMIT
000014   L$BMXV:.BLKW    1       ; TASK HIGH VIRTUAL ADDRESS LIMIT
000016   L$BLDZ:.BLKW    1       ; TASK LOAD SIZE (32W BLOCKS)
000020   L$BMXZ:.BLKW    1       ; TASK MAX. SIZE (32W BLOCKS)
000022   L$BOFF:.BLKW    1       ; TASK OFFSET INTO PARTITION
                                 ; (32W BLOCKS)
000024   L$BWND:.BLKB    1       ; NUMBER OF TASK WINDOWS
                                 ; (LESS LIBRARIES)
000025   L$BSYS:.BLKB    1       ; SYSTEM IDENTIFICATION
000026   L$BSEG:.BLKW    1       ; SIZE OF TASK SEGMENT DESCRIPTORS (BYTES
000030   L$BFLG:.BLKW    1       ; TASK FLAGS WORD
000032   L$BDAT:.BLKW    3       ; TASK CREATION DATE (YR., MO., DAY)
000040   L$BLIB:.BLKW    <7.*<R$LSIZ/2>>+1 ; RESIDENT LIBRARY ENTRIES
000346   L$BPRI:.BLKW    1       ; TASK PRIORITY
000350   L$BXFR:.BLKW    1       ; TASK TRANSFER ADDRESS
000352   L$BEXT:.BLKW    1       ; TASK EXTEND SIZE (32W BLOCKS)
000354   L$BSGL:.BLKW    1       ; RELATIVE BLOCK NUMBER OF SEGMENT
                                 ; LENGTH LIST
000356   L$BHRB:.BLKW    1       ; RELATIVE BLOCK NUMBER OF TASK
                                 ; IMAGE HEADER
000360   L$BBLK:.BLKW    1       ; NUMBER OF BLOCKS IN LABEL
000362   L$BLUN:.BLKW    1       ; NUMBER OF LOGICAL UNITS
000364   L$BROB:.BLKW 1          ; RELATIVE BLOCK NUMBER OF R/O
                                 ; TASK IMAGE
000366   L$BROL:.BLKW    1       ; R/O LOAD SIZE (32W BLOCKS)
000370   L$BRDL:.BLKW    1       ; R/O DATA LOAD SIZE (32W BLOCKS)
000372   L$BHDB:.BLKW    1       ; RELATIVE BLOCK NUMBER OF DATA HEADER
000374   L$BDHV:.BLKW    1       ; DATA WINDOW 1 HIGH VIRTUAL ADDRESS
000376   L$BDMV:.BLKW    1       ; DATA HIGH VIRTUAL ADDRESS
000400   L$BDLZ:.BLKW    1       ; DATA LOAD SIZE
000402   L$BDMZ:.BLKW    1       ; DATA MAX SIZE
000404          .BLKW    <512.-.>/2
001000   L$BASG:.BLKW    0       ; START OF DEVICE ASSIGNMENT TABLES
000340   $LBXL=<8.*<R$LSIZ>>     ; LENGTH OF EXTRA WINDOWS ADDED FOR
                                 ; M+ TASK'S

         ;
         ; OFFSETS AT END OF LABEL BLOCK
         ;

000772   .=772
000772   L$BFL2:.BLKW    1       ; SECOND TASK FLAGS WORD
000774   L$BLRL:.BLKW    1       ; LABEL BLOCK REVISION NUMBER
000776   L$AME:  .BLKW   1       ; MUST ALWAYS BE NULL
         .IIF NE .-1000  .ERROR ;DEFINITIONS OVERLAP NEXT LABEL BLOCK

;
; DEFINE LABEL BLOCK REVISION NUMBER. THIS SHOULD BE INCREMENTED
; WHENEVER A NEW FIELD IS ADDED TO THE LABEL BLOCK. IT IS
```

```
; STRUCTURED WITH MAJOR/MINOR REVISION NUMBERS IN THE HIGH/LOW
; BYTES, RESPECTIVELY. L$BLRL IS VALID ONLY IF TS$NEW IS SET.
; LABEL BLOCKS WITHOUT TS$NEW SET ARE CONSIDERED REVISION 0. THIS
; TURNS OUT HANDY, SINCE WE BELIEVE THEY ALL HAVE ZEROS IN THE
; L$BLRL OFFSET, ANYWAY.
;
         000400  LB$REV=000400  ; MAJOR REVISION LEVEL = 1
                                ; MINOR REVISION LEVEL = 0

         ;
         ; LABEL BLOCK TASK FLAG WORD DEFINITIONS
         ;

100000   TS$PIC=100000   ; TASK IS PIC (1=YES)
040000   TS$NHD=040000   ; NO HEADER IN TASK IMAGE (1=YES)
020000   TS$ACP=020000   ; TASK IS ANCILLARY CONTROL PROCESSOR (1=YES)
010000   TS$PMD=010000   ; GENERATE POST-MORTEM DUMP (1=YES)
004000   TS$SLV=004000   ; TASK IS SLAVEABLE (1=YES)
002000   TS$NSD=002000   ; NO SEND TO TASK IS PERMITTED (1=YES)
000400   TS$PRV=000400   ; TASK IS PRIVELEGED (1=YES)
000200   TS$CMP=000200   ; TASK BUILT IN COMPATIBILITY MODE (1=YES)
000100   TS$CHK=000100   ; TASK IS CHECKPOINTABLE (0=YES)
000040   TS$RES=000040   ; TASK HAS RESIDENT OVERLAYS (1=YES)
000020   TS$IOP=000020   ; PRIVILEGED TASK NOT MAPPED TO I/O PAGE
000010   TS$SUP=000010   ; TASK LINKED TO A SUPER MODE LIBRARY (1=YES)
000004   TS$XHR=000004   ; TASK HAS AN EXTERNAL HEADER (1=YES)
000002   TS$NXH=000002   ; TASK CAN NOT HAVE AN EXTERNAL HEADER (1=YES)
000001   TS$NEW=000001   ; LABEL BLOCK USES NEW FORMAT
                         ; (MEANS L$BLRL DESCRIBES FORMAT)
         ;
         ; SECOND TASK FLAGS WORD
         ;

000002   T2$FMP=000002   ; TASK USES FAST MAP DIRECTIVE (1=YES)
000001   T2$CLI=000001   ; TASK IS A CLI (1=YES)

000000          .PSECT
         .MACRO  LBLDF$  X,Y
         .ENDM
```

A.17 LNMDF$

```
;+
;         LOGICAL NAME BLOCK (LNB)
;-
001000            .ASECT
000000   .=0
000000   L.NLNK: .BLKW   1       ;LINK WORD
000002   L.NLNS: .BLKW   1       ;SIZE OF LOGICAL NAME STRING
000004   L.NENS: .BLKW   1       ;LENGTH OF EQUIVALENCE NAME STRING
000006   L.NMOD: .BLKW   1       ;MODIFIER VALUE
000010   L.NCNT: .BLKW   1       ;USE COUNT
000012   L.NCBT: .BLKW   2       ;CHANNEL BLOCK TABLE BIAS AND OFFSET
000016   L.NNAM:                 ;VARIABLE LENGTH LOGICAL NAME STRING
000016   L.NHSZ=.                ;SIZE OF LOGICAL NAME BLOCK HEADER
;+
;  TABLE NUMBER DEFINITIONS
;-
000000   LT.SYS=0                ;SYSTEM WIDE LOGICAL NAME
000001   LT.GRP=1                ;GROUP LOGICAL NAME (UIC GROUP NUMBER)
000002   LT.USR=2                ;USER SPECIFIC LOGICAL NAME
000003   LT.TSK=3                ;TASK SPECIFIC LOGICAL NAME
000004   LT.SES=4                ;SESSION LOGICAL NAME
;+
;  STATUS BITS
;-
000001   LS.TRM=1                ;TERMINAL LOGICAL
000000            .PSECT
         .MACRO  LNMDF$  X,Y,Z
         .ENDM
000001   .END
```

A.18 PCBDF$

```
;+
; MAIN PARTITION PCB
;-
                  .ASECT
        .=0
000000  P.LNK:  .BLKW   1       ;LINK TO NEXT MAIN PARTITION PCB
000002          .BLKW   1       ;(UNUSED)
000004  P.NAM:  .BLKW   2       ;PARTITION NAME IN RADIX-50
000010  P.SUB:  .BLKW   1       ;POINTER TO FIRST SUBPARTITION
000012  P.MAIN: .BLKW   1       ;POINTER TO SELF
000014  P.REL:  .BLKW   1       ;STARTING PHYSICAL ADDRESS IN 32W
                                ;BLOCKS
000016  P.BLKS:
000016  P.SIZE: .BLKW   1       ;SIZE OF PARTITION IN 32W BLOCKS
000020  P.WAIT: .BLKW   2       ;PARTITION WAIT QUEUE LISTHEAD
000024          .BLKW   2       ;(UNUSED)
000030  P.STAT: .BLKW   1       ;PARTITION STATUS FLAGS
000032  P.ST2:  .BLKW   1       ;STATUS EXTENSION FOR COMMON AND MAIN
                                ;PCBs
000034          .BLKW   3       ;(UNUSED)

000042  P.HDLN: .BLKB   1       ;SIZE OF EXTERNAL HEADER IN 32W BLOCKS
000043  P.IOC:  .BLKB   1       ;PARTITION I/O COUNT

        $$$=.

        P.RRM:  .BLKW   1       ;REQUIRED RUN MASK

                .IF NDF M$$PRO

        .=$$$

                .ENDC


                .IF NB  SYSDF

000044  P.LGTH=.                ;PARTITION CONTROL BLOCK LENGTH

                .ENDC


;+
; TASK REGION PCB
;-
        .=0
000000  P.LNK:  .BLKW   1       ;UTILITY LINK WORD
000002  P.PRI:  .BLKB   1       ;PRIORITY OF PARTITION
000003  P.RMCT: .BLKB   1       ;RESIDENT MAPPED TASKS COUNT
```

PCBDF$

```
000004    P.NAM:   .BLKW   2       ;PARTITION NAME IN RADIX-50
000010    P.SUB:   .BLKW   1       ;POINTER TO NEXT SUBPARTITION
000012    P.MAIN:  .BLKW   1       ;POINTER TO MAIN PARTITION
000014    P.REL:   .BLKW   1       ;STARTING PHYSICAL ADDR IN 32W BLOCKS
000016    P.BLKS:
000016    P.SIZE:  .BLKW   1       ;SIZE OF PARTITION IN 32W BLOCKS
000020             .BLKW   1       ;(UNUSED)
000022     .SWSZ:  .BLKW   1       ;PARTITION SWAP SIZE
000024    P.DPCB:  .BLKW   1       ;CHECKPOINT ALLOCATION PCB
000026    P.TCB:   .BLKW   1       ;TCB ADDRESS OF OWNER TASK
000030    P.STAT:  .BLKW   1       ;PARTITION STATUS FLAGS
000032    P.HDR:   .BLKW   1       ;POINTER TO HEADER CONTROL BLOCK
000034             .BLKW   1       ;(UNUSED)
000036    P.ATT:   .BLKW   2       ;ATTACHMENT DESCRIPTOR LISTHEAD

000042    P.HDLN:  .BLKB   1       ;SIZE OF EXTERNAL HEADER IN 32W BLOCKS
000043    P.IOC:   .BLKB   1       ;PARTITION I/O COUNT

          000044                   $$$=.

000044    P.RRM:.BLKW      1       ;REQUIRED RUN MASK

          .IF NDF M$$PRO

          .=$$$

          .ENDC


;+
; COMMON REGION PCB
;-
          .=0

000000    P.LNK:   .BLKW   1       ;UTILITY LINK WORD
000002    P.PRI:   .BLKB   1       ;PRIORITY OF PARTITION
000003    P.RMCT:  .BLKB   1       ;RESIDENT MAPPED TASKS COUNT
000004    P.NAM:   .BLKW   2       ;PARTITION NAME IN RADIX-50
000010    P.SUB:   .BLKW   1       ;POINTER TO NEXT SUBPARTITION
000012    P.MAIN:  .BLKW   1       ;POINTER TO MAIN PARTITION
000014    P.REL:   .BLKW   1       ;STARTING PHYSICAL ADDR IN 32W BLOCKS
000016    P.BLKS:
000016    P.SIZE:  .BLKW   1       ;SIZE OF PARTITION IN 32W BLOCKS
000020    P.CBDL:  .BLKW   1       ;COMMON BLOCK DIRECTORY LINK
000022    P.SWSZ:  .BLKW   1       ;PARTITION SWAP SIZE
000024    P.DPCB:  .BLKW   1       ;POINTER TO DISK PCB
000026    P.OWN:   .BLKW   1       ;OWNING UIC OF REGION
000030    P.STAT:  .BLKW   1       ;PARTITION STATUS FLAGS
000032    P.ST2:   .BLKW   1       ;STATUS EXTNSN FOR COMMON AND MAIN
                                   ;PCBs
000034    P.PRO:   .BLKW   1       ;PROTECTION WORD [DEWR,DEWR,DEWR,DEWR]
```

```
000036  P.ATT:  .BLKW   2       ;ATTACHMENT DESCRIPTOR LISTHEAD

000042  P.HDLN: .BLKB   1       ;SIZE OF EXTERNAL HEADER IN 32W BLOCKS
000043  P.IOC:  .BLKB   1       ;PARTITION I/O COUNT

        $$$=.

000044  P.RRM:.BLKW     1       ;REQUIRED RUN MASK

                .IF NDF M$$PRO

        .=$$$

        .ENDC

        .PSECT

;+
; PARTITION STATUS WORD BIT DEFINITIONS
;-

        PS.OUT=100000           ;PARTITION IS OUT OF MEMORY(1=YES)
        PS.CKP=40000            ;PARTITION CHECKPOINT IN PROGRESS
                                ;(1=YES)
        PS.CKR=20000            ;PARTITION CHECKPOINT IS REQUESTED
                                ;(1=YES)
        PS.CHK=10000            ;PARTITION IS NOT CHECKPOINTABLE
                                ;(1=YES)
        PS.FXD=4000             ;PARTITION IS FIXED (1=YES)
        PS.CAF=2000             ;CHECKPOINT SPACE ALLOCATION FAILURE
                                ;(1=YES)
        PS.LIO=1000             ;MARKED BY SHUFFLER FOR LONG I/O
                                ;(1=YES)
        PS.NSF=400              ;PARTITION IS NOT SHUFFLEABLE (1=YES)
        PS.COM=200              ;LIBRARY OR COMMON BLOCK (1=YES)
        PS.LFR=100              ;LAST LOAD OF REGION FAILED (1=YES)
        PS.PER=40               ;PARTIY ERROR OCCURED IN THIS REGION
                                ;(1=YES)
        PS.DEL=10               ;PARTITION SHOULD BE DELETED WHEN NOT
                                ;ATTACHED (1=YES)
        PS.AST=4                ;PARTITION HAS REGION LOAD AST PENDING

;+
; REQUIRED RUN MASK
;-

        PR.UBT=100000           ;UNIBUS RUN T
        PR.UBS=40000            ;UNIBUS RUN S
        PR.UBR=20000            ;UNIBUS RUN R
        PR.UBP=10000            ;UNIBUS RUN P
        PR.UBN=4000             ;UNIBUS RUN N
```

```
         PR.UBM=2000             ;UNIBUS RUN M
         PR.UBL=1000             ;UNIBUS RUN
         PR.UBK=400              ;UNIBUS RUN K
         PR.UBJ=200              ;UNIBUS RUN J
         PR.UBH=100              ;UNIBUS RUN H
         PR.UBF=40               ;UNIBUS RUN F
         PR.UBE=20               ;UNIBUS RUN E
         PR.CPD=10               ;PROCESSOR  D
         PR.CPC=4                ;PROCESSOR  C
         PR.CPB=2                ;PROCESSOR
         PR.CPA=1                ;PROCESSOR  A

;+
; STATUS EXTENSION WORD BIT DEFINITIONS
;        (THESE BITS CAN ONLY BE EXAMINED IN COMMON OR MAIN
;        PCBs)
;-

P2.LMA=40000      ;DON'T SHUFFLE,DELETE SPINDLE OR MUTILATE
                  ;THIS PARTITION (ACTUALLY ON P/OS V1.7 AND V2.0
                  ;THIS BIT HAS TAKEN ON THE EXACT OPPOSITE
                  ;MEANING SINCE IT HAS YET TO BE IMPLEMENTED ON
                  ;M-PLUS. TEMPORARILY, IT HAS BEEN REDEIFNED TO
                  ;MEAN  "THIS COMMON IS PART OF THE APPL.
                  ;AND SHOULD BE REMOVED FROM THE SYSTEM (IF
                  ;POSSIBLE) WHEN THE APPLICATION EXITS")
P2.CPC=20000      ;CPCR INITIATED CHECKPOINT PENDING
P2.SEC=4000       ;THIS IS RO SECTION OF MU TASK WITH TCB IN SEC. POOL
P2.PAR=2000       ;THE FIXER TASK HAS HANDLED A PARITY ERROR
P2.POL=1000       ;SECONDARY POOL PARTITION
P2.CPU=400        ;MULTIPROCESSOR CPU PARTITION
P2.PIC=200        ;POSITION INDEPENDENT LIBRARY OR COMMON (1=YES)
P2.RON=100        ;READ-ONLY COMMON (1=YES)
P2.DRV=40         ;DRIVER COMMON PARTITION (1=YES)
P2.APR=7 ;STARTING APR NUMBER MASK FOR NON-PIC COMMON

;+
; CHECKPOINT FILE PCB
;-

                  .ASECT
         .=0
000000   P.LNK:   .BLKW   1       ;LINK WORD OF CHECKPOINT FILE PCBs
000002   P.UCB:   .BLKW   1       ;UCB ADDRESS OF CHECKPOINT FILE DEVICE
000004   P.LBN:   .BLKW   1       ;HIGH PART OF STARTING LBN
000006            .BLKW   1       ;LOW PART OF STARTING LBN
000010   P.SUB:   .BLKW   1       ;POINTER TO FIRST CHECKPOINT
                                  ;ALLOCATION PCB
000012   P.MAIN:  .BLKW   1       ;MUST BE 0 (FOR $RLPR1)
000014   P.REL:   .BLKW   1       ;CONTAINS 0 IF FILE IN USE, 1 IF NOT
                                  ;IN USE
```

```
000016   P.SIZE: .BLKW   1      ;SIZE OF CHECKPOINT FILE IN 256W
                                ;BLOCKS
000020   P.DLGH=.               ;LENGTH OF ALL DISK PCBs

;+
; CHECKPOINT ALLOCATION PCB
;-

         .=0
000000           .BLKW   4      ;(UNUSED)
000010   P.SUB:  .BLKW   1      ;LINK TO NEXT CHECKPOINT ALLOCATION
                                ;PCB
000012   P.MAIN: .BLKW   1      ;ADDRESS OF CHECKPOINT FILE PCB
000014   P.REL:  .BLKW   1      ;RELATIVE POSITION IN FILE IN 256W
                                ;BLOCKS
000016   P.SIZE: .BLKW   1      ;SIZE ALLOCATED IN 256W BLOCKS

;+
; COMMON TASK IMAGE FILE PCB
;-

         .=0
000000   P.FID1: .BLKW   1      ;FILE ID WORD FOR SAVE
000002   P.UCB:  .BLKW   1      ;UCB ADDRESS OF DEVICE ON WHICH
                                ;COMMON RESIDES
000004   P.LBN:  .BLKW   1      ;HIGH PART OF STARTING LBN
000006           .BLKW   1      ;LOW PART OF STARTING LBN
000010   P.FID2: .BLKW   1      ;FILE ID WORD FOR SAVE
000012   P.MAIN: .BLKW   1      ;POINTER TO SELF
000014   P.REL:  .BLKW   1      ;ALWAYS CONTAINS A 0
000016   P.FID3: .BLKW   1      ;FILE ID WORD FOR SAVE

;+
; ATTACHMENT DESCRIPTOR OFFSETS
;-
         .ASECT
         .=0
000000   A.PCBL: .BLKW   1      ;PCB ATTACHMENT QUEUE THREAD WORD
000002   A.PRI:  .BLKB   1      ;PRIORITY OF ATTACHED TASK
000003   A.IOC:  .BLKB   1      ;I/O COUNT THROUGH THIS DESCRIPTOR
000004   A.TCB:  .BLKW   1      ;TCB ADDRESS OF ATTACHED TASK
000006   A.TCBL: .BLKW   1      ;TCB ATTACHMENT QUEUE THREAD WORD
000010   A.STAT: .BLKB   1      ;STATUS BYTE
000011   A.MPCT: .BLKB   1      ;MAPPING COUNT OF TASK THRU THIS
                                ;DESCRIPTOR
000012   A.PCB:  .BLKW   1      ;PCB ADDRESS OF ATTACHED TASK
000014   A.LGTH= .              ;LENGTH OF ATTACHMENT DESCRIPTOR

;+
; ATTACHMENT DESCRIPTOR STATUS BYTE BIT DEFINITIONS
;-
```

PCBDF$

```
.PSECT
AS.PRO=100        ;A.TCB IS SEC POOL PROTO TCB BIAS (1=YES)
AS.SBP=20         ;CACHE BYPASS REQUESTED
AS.RBP=40         ;REQUEST TO NOT BYPASS CACHE
AS.DEL=10         ;TASK HAS DELETE ACCESS (1=YES)
AS.EXT=4          ;TASK HAS EXTEND ACCESS (1=YES)
AS.WRT=2          ;TASK HAS WRITE ACCESS (1=YES)
AS.RED=1          ;TASK HAS READ ACCESS (1=YES)
```

A.19 PKTDF$

```
;+
; ASYNCHRONOUS SYSTEM TRAP CONTROL BLOCK OFFSET DEFINITIONS
;
; SOME POSITIONAL DEPENDENCIES BETWEEN THE OCB AND THE AST CONTROL
; BLOCK ARE RELIED UPON IN THE ROUTINE $FINXT IN THE MODULE SYSXT.
;-

                 .ASECT
         .=177774
177774   A.KSR5: .BLKW   1       ;SUBROUTINE KISAR5 BIAS (A.CBL=0)
177776   A.DQSR: .BLKW   1       ;DEQUEUE SUBROUTINE ADDRESS (A.CBL=0)
000000           .BLKW   1       ;AST QUEUE THREAD WORD
000002   A.CBL:  .BLKW   1       ;LENGTH OF CONTROL BLOCK IN BYTES
                                 ;IF A.CBL = 0, THE AST CONTROL BLOCK IS
                                 ;TO BE DEALLOCATED BY THE DEQUEUE
                                 ;SUBROUTINE POINTED TO BY A.DQSR
                                 ;MAPPED VIA APR 5 VALUE A.KSR5.  THIS
                                 ;IS CURRENTLY USED ONLY BY THE FULL
                                 ;DUPLEX TERMINAL DRIVER FOR UNSOLICITED
                                 ;CHARACTER ASTS. IF THE LOW BYTE OF
                                 ;A.CBL = 0, AND THE HIGH BYTE IS NOT
                                 ;= 0, THE AST CONTROL BLOCK IS A
                                 ;SPECIFIED AST, WITH LENGTH, C.LGTH.
                                 ;IF THE HIGH BYTE OF A.CBL=0
                                 ;AND THE LOW BYTE > 0, THEN
                                 ;THE LOW BYTE IS THE LENGTH OF THE
                                 ;AST CONTROL BLOCK.
                                 ;IF HIGH BYTE = 0 AND LOW BYTE IS
                                 ;NEGATIVE, THEN THE BLOCK IS A KERNEL
                                 ;AST BIT 6 IS SET IF $SGFIN SHOULD
                                 ;NOT BE CALLED PRIOR TO DISPATCHING
                                 ;THE AST, AND THE LOW SIX BITS (5-0)
                                 ;REPRESENT THE INDEX/2 INTO THE
                                 ;KERNEL AST DISPATCH TABLE ($KATBL)
000004   A.BYT:  .BLKW   1       ;NUMBER OF BYTES TO ALLOCATE ON TASK
                                 ;STACK
000006   A.AST:  .BLKW   1       ;AST TRAP ADDRESS
000010   A.NPR:  .BLKW   1       ;NUMBER OF AST PARAMETERS
000012   A.PRM:  .BLKW   1       ;FIRST AST PARAMETER
         AS.FPA=1                ;CODE FOR FLOATING POINT AST
         AS.RCA=2                ;CODE FOR RECEIVE DATA AST
         AS.RRA=3                ;CODE FOR RECEIVE BY REFERENCE AST
         AS.PEA=4                ;CODE FOR PARITY ERROR AST
         AS.REA=5                ;CODE FOR REQUESTED EXIT AST
         AS.PFA=6                ;CODE FOR POWER FAIL AST
         AS.CAA=7                ;CODE FOR CLI COMMAND ARRIVAL AST
         AS.TEA=10               ;CODE FOR TAST EXIT AST
         ;
         ; ABORTER SUBCODES FOR ABORT AST (AS.REA) TO BE RETURNED ON
```

```
        ; USER'S STACK
        ;
        AB.NPV=1        ;ABORTER IS NONPRIVILEGED (1=YES)
        AB.TYP=2        ;ABORT FROM DIRECTIVE (0=YES)
                        ;ABORT FROM CLI COMMAND (1=YES)
        A.PLGH=70       ;SIZE OF PARITY ERROR AST CONTROL BLOCK
        A.DUCB=10       ;UCB OF TERM ISSUING DEBUG COMMAND
        A.DLGH=10.      ;LENGTH OF DEBUG (AK.TBT) AST BLOCK

        ; KERNEL AST CONTROL CODES (A.CBL)

        AK.BUF=200      ;BUFFERED I/O COMPLETION
                        ;THIS CODE MUST BE 200 UNTIL ALL
                        ;REFERENCES IN TTDRV ARE FIXED
        AK.OCB=201      ;OFFSPRING TASK EXIT
        AK.GBI=202      ;SEGMENTED BUFFERED I/O COMPLETION
        AK.TBT=203      ;TASK FORCE T-BIT TRAP (DEBUG CMD)
        AK.DIO=204      ;DELAYED I/O COMPLETION
        AK.GGF=205      ;GRP. GBL. RUNDWN

;+
; I/O PACKET OFFSET DEFINITIONS
;-

                .ASECT
        .=0
000000  I.LNK:  .BLKW   1       ;I/O QUEUE THREAD WORD
000002  I.PRI:  .BLKB   1       ;REQUEST PRIORITY
000003  I.EFN:  .BLKB   1       ;EVENT FLAG NUMBER
000004  I.TCB:  .BLKW   1       ;TCB ADDRESS OF REQUESTOR
000006  I.LN2:  .BLKW   1       ;POINTER TO SECOND LUN WORD
000010  I.UCB:  .BLKW   1       ;POINTER TO UNIT CONTROL BLOCK
000012  I.FCN:  .BLKW   1       ;I/O FUNCTION CODE
000014  I.IOSB: .BLKW   1       ;VIRTUAL ADDRESS OF I/O STATUS BLOCK
000016          .BLKW   1       ;I/O STATUS BLOCK RELOCATON BIAS
000020          .BLKW   1       ;I/O STATUS BLOCK ADDRESS
000022  I.AST:  .BLKW   1       ;AST SERVICE ROUTINE ADDRESS
000024  I.PRM:  .BLKW   1       ;RESERVED FOR MAPPING PARAMETER #1
000026          .BLKW   6       ;PARAMETERS 1 TO 6
000042          .BLKW   1       ;USER MODE DIAGNOSTIC PARAMETER WORD
000044  I.ATTL=.                ;MINIMUM LENGTH OF I/O PACKET (USED BY
                                ;FILE SYSTEM TO CALCULATE MAXIMUM
                                ;NUMBER OF ATTRIBUTES)
        I.AADA: .BLKW   2       ;STORAGE FOR ATT DESCR PTRS WITH I/O
000050  I.LGTH=.                ;LENGTH OF I/O REQUEST CONTROL BLOCK
        I.ATRL=6*8.             ;LENGTH OF FILE SYSTEM ATTRIBUTE BLOCK

;+
; ANCILLARY CONTROL BLOCK (ACB) DEFINITIONS
;-
```

```
        .=0
000000  A.REL:  .BLKW   1       ;ACD RELOCATION BIAS
000002  A.DIS:  .BLKW   1       ;ACD DISPATCH TABLE POINTER
000004  A.MAS:  .BLKW   1       ;ACD FUNCTION MASK
000006  A.NUM:  .BLKB   1       ;ACD IDENTIFICATION NUMBER
000007          .BLKB   1       ;RESERVED
000010  A.LIN:  .BLKW   1       ;ACD LINK WORD
000012  A.ACC:  .BLKB   1       ;ACD ACCESS COUNT
000013  A.STA:  .BLKB   1       ;ACD STATUS BYTE
000014  A.LEN1=.                ;LENGTH OF PROTOTYPE ACB
        ;
        .=A.LIN                 ;FULL ACB OVERLAPS PROTOTYPE ACB
000010  A.IMAP: .BLKW   1       ;ACD INTERRUPT BUFFER RELOCATION BIAS
000012  A.IBUF: .BLKW   1       ;ACD INTERRUPT BUFFER ADDRESS
000014  A.ILEN: .BLKW   1       ;ACD INTERRUPT BUFFER LENGTH
000016  A.SMAP: .BLKW   1       ;ACD SYSTEM STATE BUFFER RELOCATION
                                ;BIAS
000020  A.SBUF: .BLKW   1       ;ACD SYSTEM STATE BUFFER ADDRESS
000022  A.SLEN: .BLKW   1       ;ACD SYSTEM STATE BUFFER LENGTH
000024  A.IOS:  .BLKW   2       ;ACD I/O STATUS
000030  A.RES:  .BLKW   2       ;RESERVED FOR USE BY THE ACD
000034  A.LEN2=.                ;LENGTH OF FULL ACB
                                ;
                                ; DEFINE THE FLAG VALUES IN THE OFFSET
                                ; U.AFLG
                                ;
        UA.ACC=1                ;ACCEPT THIS CHARACTER
        UA.PRO=2                ;PROCESS THIS CHARACTER
        UA.ECH=4                ;ECHO THIS CHARACTER
        UA.TYP=10               ;FORCE THIS CHARACTER INTO TYPEAHEAD
        UA.SPE=20               ;THIS CHARACTER HAS A SPECIAL ECHO
        UA.PUT=40               ;PUT THIS CHARACTER IN THE INPUT BUFFER
        UA.CAL=100              ;CALL THE ACD BACK AFTER THE TRANSFER
        UA.COM=200              ;COMPLETE THE INPUT REQUEST
                                ;
        UA.ALL=400              ;ALLOW PROCESSING OF THIS I/O REQUEST
        UA.TRA=1000             ;TRANSFER CHARS. WHEN I/O COMPLETES
        ;
        ; DEFINE THE ACD ENTRY POINTS (OFFSETS INTO THE DISPATCH TABLE)
        ;
        .=0
000000  A.ACCE: .BLKW   1       ;I/O REQUEST ACCEPTANCE ENTRY POINT
000002  A.DEQU: .BLKW   1       ;I/O REQUEST DEQUEUE ENTRY POINT
000004  A.POWE: .BLKW   1       ;POWER FAILURE ENTRY POINT
000006  A.INPU: .BLKW   1       ;INPUT COMPLETION ENTRY POINT
000010  A.OUTP: .BLKW   1       ;OUTPUT COMPLETION ENTRY POINT
000012  A.CONN: .BLKW   1       ;CONNECTION ENTRY POINT
000014  A.DISC: .BLKW   1       ;DISCONNECTION ENTRY POINT
000016  A.RECE: .BLKW   1       ;INPUT CHARACTER RECEPTION ENTRY POINT
000020  A.PROC: .BLKW   1       ;INPUT CHARACTER PROCESSING ENTRY POINT
000022  A.CALL: .BLKW   1       ;CALL ACD BACK AFTER TRANSFER ENTRY
```

PKTDF$

```
                                      ;POINT
        ;
        ; DEFINE THE STATUS BITS IN A.STA OF THE PROTOTYPE ACB
        ;
000001  AS.DEL=1                      ;ACD IS MARKED FOR DELETE
000002  AS.DIS=2                      ;ACD IS DISABLED

;+
; SECONDARY POOL COMMAND BUFFER BLOCKS
;-
        .=0
000000  C.CLK:  .BLKW   1             ;LINK WORD
000002  C.CTCB: .BLKW   1             ;TCB ADDRESS OF TASK TO RECEIVE COMMAND
000004  C.CUCB: .BLKW   1             ;UCB ADDRESS OF RESPONSIBLE TERMINAL
000006  C.CCT:  .BLKW   1             ;CHARACTER COUNT, EXCLUDING TRAILING
                                      ;CR
000010  C.CSTS: .BLKW   1             ;STATUS MASK
000012  C.CMCD:                       ;SYSTEM MESSAGE CODE
000012  C.CSO:  .BLKW   1             ;STARTING OFFSET OF VALID COMMAND
                                      ;TEXT
000014  C.CTR:  .BLKB   1             ;TERMINATOR CHARACTER
000015  C.CBLK: .BLKB   1             ;SIZE OF PACKET IN SEC POOL (32 WD.)
                                      ;BLOCKS
000016  C.CTXT:                       ;COMMAND TEXT, FOLLOWED BY CR

;+
; BIT DEFINITIONS FOR THE GIN$ (AKA WIMP$) INFORMATION
; DIRECTIVE.
;-

        SF.PRV=100000   ;FUNCTION IS PRIVILEGED
        SF.IN= 40000    ;FUNCTION IS AN INPUT FUNCTION

;+
; OFFSPRING CONTROL BLOCK DEFINITIONS
;
; SOME POSITIONAL DEPENDENCIES ARE DEPENDED ON BETWEEN THE
; OCB AND THE AST BLOCK IN THE ROUTINE $FINXT IN THE MODULE
; SYSXT.
;-
        .=0
000000  O.LNK:  .BLKW   1             ;OCB LINK WORD
000002  O.MCRL: .BLKW   1             ;ADDRESS OF MCR COMMAND LINE
000004  O.PTCB: .BLKW   1             ;PARENT TCB ADDRESS
000006  O.AST:  .BLKW   1             ;EXIT AST ADDRESS
000010  O.EFN:  .BLKW   1             ;EXIT EVENT FLAG
000012  O.ESB:  .BLKW   1             ;EXIT STATUS BLOCK VIRTUAL ADDRESS
000014  O.STAT: .BLKW   8.            ;EXIT STATUS BUFFER

000034  O.LGTH=.                      ;LENGTH OF OCB
```

PKTDF$

```
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; THE FOLLOWING CPB,C.PSTS,AND C.CMCD ARE NOT CURRENTLY USED BY P/OS.
; THEY ARE, HOWEVER, RESERVED FOR A POSSIBLE FUTURE USE.
;-------------------------------------------------------------------------

         ;
         ; CLI PARSER BLOCK (CPB) DEFINITIONS
         ;
         .=0
000000   C.PTCB: .BLKW   1        ;ADDRESS OF CLI'S TCB
000002   C.PNAM: .BLKW   2        ;CLI NAME
000006   C.PSTS: .BLKW   1        ;STATUS MASK
000010   C.PDPL: .BLKB   1        ;LENGTH OF DEFAULT PROMPT
000011   C.PCPL: .BLKB   1        ;LENGTH O CNTRL/C PROMPT
000012   C.PRMT:                  ;START OF PROMPT STRINGS. DEFAULT
                                  ;IS CONCATENATED WITH CONTROL C PROMPT

         ;
         ; STATUS BIT DEFINITIONS
         ;
         CP.NUL=1                 ;PASS EMPTY COMMANDS TO CLI
         CP.MSG=2                 ;CLI DESIRES SYSTEM MESSAGES
         CP.LGO=4                 ;CLI WANTS COMMANDS FROM LOGGED OFF
                                  ;TTYS
         CP.DSB=10                ;CLI IS DISABLED
         CP.PRV=20                ;USER MUST BE PRIV TO SET TTY TO THIS
                                  ;CLI
         CP.SGL=40                ;DON'T HANDLE CONTINUATIONS (M-PLUS
                                  ;ONLY)
         CP.NIO=100               ;MCR..., HEL, BYE DO NO I/O TO TTY
                                  ;HEL, BYE DO NOT SET CLI ETC.
         CP.RST=200               ;ABILITY TO SET TO THIS CLI IS
                                  ;RESTRICTED
                                  ;TO THE CLI ITSELF
         CP.EXT=400               ;PASS TASK EXIT PROMPT REQUESTS TO CLI
         CP.POL=1000              ;CLI TCB IS IN SECONDARY POOL
         CP.CTC=2000              ;^C NOTIFICATION PACKETS ARE WANTED

                 ;
                 ; STATUS BITS FOR COMMAND BLOCKS
                 ;
                 CC.MCR=1         ;FORCE COMMAND TO MCR
                 CC.PRM=2         ;ISSUE DEFAULT PROMPT
                 CC.EXT=4         ;TASK EXIT PROMPT REQUEST
                 CC.KIL=10        ;DELETE ALL CONTINUATION PIECES FROM
                                  ;THIS TTY
                 CC.CLI=20        ;COMMAND TO BE RETREIVED BY GCCI$ ONLY
                 CC.MSG=40        ;PACKET CONTAINS SYSTEM MESSAGE TO CLI
                 CC.TTD=100       ;COMMAND CAME FROM TTDRV
                 CC.CTC=200       ;^C NOTIFICATION PACKET
```

```
                ;
                ; IDENTIFIER CODES FOR SYSTEM TO CLI MESSAGES
                ;
                ; CODES 0-127. ARE RESERVED FOR USE BY DIGITAL
                ; CODES 128.-255. ARE RESERVED FOR USE BY CUSTOMERS
                ;
                CM.INE=1        ;CLI INITIALIZED ENABLED
                CM.IND=2        ;CLI INITIALIZED DISABLED
                CM.CEN=3        ;CLI ENABLED
                CM.CDS=4        ;CLI DISABLED
                CM.ELM=5        ;CLI BEING ELIMINATED
                CM.EXT=6        ;CLI MUST EXIT IMMEDIATELY
                CM.LKT=7        ;NEW TERMINAL LINKED TO CLI
                CM.RMT=8.       ;TERMINAL REMOVED FROM CLI
                CM.MSG=9.       ;GENERAL MESSAGE TO CLI

        ;+
        ; GROUP GLOBAL EVENT FLAG BLOCK OFFSETS
        ; (CURRENTLY NOT USED BY P/OS)
        ;-
                .=0
000000  G.LNK:  .BLKW   1       ;LINK WORD
000002  G.GRP:  .BLKB   1       ;GROUP NUMBER
000003  G.STAT: .BLKB   1       ;STATUS BYTE
000004  G.CNT:  .BLKW   1       ;ACCESS COUNT
000006  G.EFLG: .BLKW   2       ;EVENT FLAGS

000012  G.LGTH=.                ;LENGTH OF GROUP GLOBAL EVENT FLAG
                                ;BLOCK

        GS.DEL=1                ;STATUS BIT -- MARKED FOR DELETE

;+
; EXECUTIVE POOL MONITOR CONTROL FLAGS (HISTORICAL INTEREST
; ONLY)
;-

; $POLST IS THE SYNCHRONIZATION WORD BETWEEN THE EXEC AND POOL
; MONITOR
;
        PC.HIH=1                ;HIGH POOL LIMIT CROSSED (1=YES)
        PC.LOW=2                ;LOW POOL LIMIT CROSSED (1=YES)
        PC.ALF=4                ;POOL ALLOCATION FAILURE (1=YES)
        PC.XIT=200              ;FORCE POOL MONITOR TASK TO EXIT (MUST
                                ;BE COUPLED WITH SETTING FE.MXT IN THE
                                ;FEATURE MASK)
        PC.NRM=PC.HIH*400       ;POOL TASK INHIBIT BIT FOR HIGH POOL
        PC.ALM=PC.LOW*400       ;POOL TASK INHIBIT BIT FOR LOW POOL

                                ;$POLFL IS THE POOL USAGE CONTROL WORD
```

```
                  PKTDF$

PF.INS=40         ;REJECT NONPRIVILEGED INS/RUN/REM
PF.LOG=100        ;NONPRIVILEGED LOGINS ARE DISABLED
PF.REQ=200        ;STALL REQUEST OF NONPRIV. TASKS

PF.ALL=177777     ;TAKE ALL POSSIBLE ACTIONS TO SAVE POOL

        .PSECT
```

A.20 QIOSY$

```
;
; SYSTEM STANDARD CODES, USED BY EXECUTIVE AND DRIVERS
;

      DECIMAL    OCTAL

IE.BAD   -01.    177777  Bad parameters
IE.IFC   -02.    177776  Invalid function code
IE.DNR   -03.    177775  Device not ready
IE.VER   -04.    177774  Parity error on device
IE.ONP   -05.    177773  Hardware option not present
IE.SPC   -06.    177772  Illegal user buffer
IE.DNA   -07.    177771  Device not attached
IE.DAA   -08.    177770  Device already attached
IE.DUN   -09.    177767  Device not attachable
IE.EOF   -10.    177766  End of file detected
IE.EOV   -11.    177765  End of volume detected
IE.WLK   -12.    177764  Write attempted to locked unit
IE.DAO   -13.    177763  Data overrun
IE.SRE   -14.    177762  Send/receive failure
IE.ABO   -15.    177761  Request terminated
IE.PRI   -16.    177760  Privilege violation
IE.RSU   -17.    177757  Sharable resource in use
IE.OVR   -18.    177756  Illegal overlay request
IE.BYT   -19.    177755  Odd byte count (or virtual address)
IE.BLK   -20.    177754  Logical block number too large
IE.MOD   -21.    177753  Invalid UDC module #
IE.CON   -22.    177752  UDC connect error
IE.BBE   -56.    177710  Bad block on device
IE.STK   -58.    177706  Not enough stack space (FCS or FCP)
IE.FHE   -59.    177705  Fatal hardware error on device
IE.EOT   -62.    177702  End of tape detected
IE.OFL   -65.    177677  Device off line
IE.BCC   -66.    177676  Block check, CRC, or framing error
IE.NFW   -69.    177673  Path lost to partner ;THIS CODE MUST BE ODD
IE.DIS   -69.    177673  Path lost to partner ;DISCONNECTED (SAME
                                AS NFW)
IE.NDR   -72.    177670  No dynamic space available ; SEE ALSO IE.UPN
IE.TMO   -95.    177641  Timeout on request      ; see also IS.TMO
IE.CNR   -96.    177640  Connection rejected
IE.MII   -99.    177635  Media inserted incorrectly
IE.SPI   -100.   177634  Spindown ignored


;
; FILE PRIMITIVE CODES
;

IE.NOD   -23.    177751  Caller's nodes exhausted
```

```
IE.DFU   -24.    177750  Device full
IE.IFU   -25.    177747  Index file full
IE.NSF   -26.    177746  No such file
IE.LCK   -27.    177745  Locked from read/write access
IE.HFU   -28.    177744  File header full
IE.WAC   -29.    177743  Accessed for write
IE.CKS   -30.    177742  File header checksum failure
IE.WAT   -31.    177741  Attribute control list format error
IE.RER   -32.    177740  File processor device read error
IE.WER   -33.    177737  File processor device write error
IE.ALN   -34.    177736  File already accessed on LUN
IE.SNC   -35.    177735  File ID, file number check
IE.SQC   -36.    177734  File ID, sequence number check
IE.NLN   -37.    177733  No file accessed on LUN
IE.CLO   -38.    177732  File was not properly closed
IE.DUP   -57.    177707  ENTER - duplicate entry in directory
IE.BVR   -63.    177701  Bad version number
IE.BHD   -64.    177700  Bad file header
IE.EXP   -75.    177665  File expiration date not reached
IE.BTF   -76.    177664  Bad tape format
IE.ALC   -84.    177654  Allocation failure
IE.ULK   -85.    177653  Unlock error
IE.WCK   -86.    177652  Write check failure
IE.DSQ   -90.    177646  Disk quota exceeded

;
; FILE CONTROL SERVICES CODES
;

IE.NBF   -39.    177731  OPEN - no buffer space available for file
IE.RBG   -40.    177730  Illegal record size
IE.NBK   -41.    177727  File exceeds space allocated, no blocks
IE.ILL   -42.    177726  Illegal operation on file descriptor block
IE.BTP   -43.    177725  Bad record type
IE.RAC   -44.    177724  Illegal record access bits set
IE.RAT   -45.    177723  Illegal record attributes bits set
IE.RCN   -46.    177722  Illegal record number - too large
IE.2DV   -48.    177720  Rename - 2 different devices
IE.FEX   -49.    177717  Rename - new file name already in use
IE.BDR   -50.    177716  Bad directory file
IE.RNM   -51.    177715  Can't rename old file system
IE.BDI   -52.    177714  Bad directory syntax
IE.FOP   -53.    177713  File already open
IE.BNM   -54.    177712  Bad file name
IE.BDV   -55.    177711  Bad device name
IE.NFI   -60.    177704  File ID was not specified
IE.ISQ   -61.    177703  Illegal sequential operation
IE.NNC   -77.    177663  Not ANSI 'D' format byte count

;
; NETWORK ACP, PSI, AND DECDATAWAY CODES
```

QIOSY$

```
;

IE.NNN   -68.    177674  No such node
IE.BLB   -70.    177672  Bad logical buffer
IE.URJ   -73.    177667  Connection rejected by user
IE.NRJ   -74.    177666  Connection rejected by network
IE.NDA   -78.    177662  No data available
IE.IQU   -91.    177645  Inconsistent qualifier usage
IE.RES   -92.    177644  Circuit reset during operation
IE.TML   -93.    177643  Too many links to task
IE.NNT   -94.    177642  Not a network task
IE.UKN   -97.    177637  Unknown name

;
; ICS/ICR ERROR CODES
;
IE.NLK   -79.    177661  Task not linked to specified ICS/ICR interrupts
IE.NST   -80.    177660  Specified task not installed
IE.FLN   -81.    177657  Device offline when offline request was issued


;
; TTY ERROR CODES
;

IE.IES   -82.    177656  Invalid escape sequence
IE.PES   -83.    177655  Partial escape sequence


;
; RECONFIGURATION CODES
;

IE.ICE   -47.    177721  Internal consistancy error
IE.ONL   -67.    177675  Device online
IE.SZE   -98.    177636  Unable to size device

;
; PCL ERROR CODES
;

IE.NTR   -87.    177651  Task not triggered
IE.REJ   -88.    177650  Transfer rejected by receiving CPU
IE.FLG   -89.    177647  Event flag already specified


;
; SUCCESSFUL RETURN CODES---
;

IS.PND   +00.    0       OPERATION PENDING
```

```
IS.SUC   +01.    1       OPERATION COMPLETE, SUCCESS
IS.RDD   +02.    2       FLOPPY DISK SUCCESSFUL COMPLETION
                                OF A READ PHYSICAL, AND DELETED
                                DATA MARK WAS SEEN IN SECTOR HEADER
IS.TNC   +02.    2       (PCL) SUCCESSFUL TRANSFER BUT MESSAGE
                                TRUNCATED (RECEIVE BUFFER TOO SMALL).
IS.CHW   +04.    4       (IBM COMM) DATA READ WAS RESULT OF
                                IBM HOST CHAINED WRITE OPERATION
IS.BV    +05.    5       (A/D READ) AT LEAST ONE BAD VALUE
                                WAS READ (REMAINDER MAY BE GOOD).
                                BAD CHANNEL IS INDICATED BY A
                                NEGATIVE VALUE IN THE BUFFER.
IS.DAO   +02.    2       SUCCESSFUL BUT WITH DATA OVERRUN
                                (NOT TO BE CONFUSED WITH IE.DAO)


;
; TTY SUCCESS CODES
;

IS.CR    <015*400+1>     CARRIAGE RETURN WAS TERMINATOR
IS.ESC   <33*400+1>      ESCAPE (ALTMODE) WAS TERMINATOR
IS.CC    <3*400+1>       CTRL/C WAS TERMINATOR
IS.ESQ   <0233*400+1>    ESCAPE SEQUENCE WAS TERMINATOR
IS.PES   <200*400+1>     PARTIAL ESCAPE SEQUENCE TERMINATOR
IS.EOT   <4*400+1>       EOT WAS TERMINATOR (BLOCK MODE INPUT)
IS.TAB   <11*400+1>      TAB WAS TERMINATOR (FORMS MODE INPUT)
IS.TMO   +2.     2       REQUEST TIMED OUT

;
; Professional Bisync Success Codes
;
IS.RVI   +2.     2        DATA SUCC. XMITTED; HOST ACKED W/RVI
IS.CNV   +3.     3        DATA SUCC. XMITTED; HOST ACKED W/CONVERSATION
IS.XPT   +5.     5        DATA SUCC. RECVD IN TRANSPARENT MODE
;
; Professional Bisync Abort Codes
;
; These codes are returned in the high byte of the first word of the
; IOSB when the low byte contains IE.ABO.
;
SB.KIL   -1.     377      ABORTED BY IO.KIL
SB.ACK   -2.     376      ABORTED BECAUSE TOO MANY ACKS RECD OUT OF SEQ
SB.NAK   -3.     375      ABORTED BECAUSE NAK THRESHOLD EXCEEDED
SB.ENQ   -4.     374      ABORTED BECAUSE ENQ THRESHOLD EXCEEDED
SB.BOF   -5.     373      ABORTED BECAUSE OF IO.RLB BUFFER OVERFLOW
SB.TMO   -6.     372      ABORTED BECAUSE OF TIMEOUT
SB.DIS   -7.     371      ABORTED BECAUSE HOST DISCONNECTED W/ DLE, EOT

;
; STANDARD ERROR CODES RETURNED BY DIRECTIVES IN THE DIRECTIVE
```

QIOSY$

```
; STATUS WORD
;
IE.UPN   -01.    177777  Insufficient dynamic storage ; SEE ALSO
                                IE.NDR
IE.INS   -02.    177776  Specified task not installed
IE.PTS   -03.    177775  Partition too small for task
IE.UNS   -04.    177774  Insufficient dynamic storage for send
IE.ULN   -05.    177773  Un-assigned LUN
IE.HWR   -06.    177772  Device handler not resident
IE.ACT   -07.    177771  Task not active
IE.ITS   -08.    177770  Directive inconsistent with task state
IE.FIX   -09.    177767  Task already fixed/unfixed
IE.CKP   -10.    177766  Issuing task not checkpointable
IE.TCH   -11.    177765  Task is checkpointable
IE.RBS   -15.    177761  Receive buffer is too small
IE.PRI   -16.    177760  Privilege violation
IE.RSU   -17.    177757  Resource in use
IE.NSW   -18.    177756  No swap space available
IE.ILV   -19.    177755  Illegal vector specified
IE.ITN   -20.    177754  Invalid table number
IE.LNF   -21.    177753  Logical name not found
;
;
;
IE.AST   -80.    177660  Directive issued/not issued from AST
IE.MAP   -81.    177657  Illegal mapping specified
IE.IOP   -83.    177655  Window has I/O in progress
IE.ALG   -84.    177654  Alignment error
IE.WOV   -85.    177653  Address window allocation overflow
IE.NVR   -86.    177652  Invalid region ID
IE.NVW   -87.    177651  Invalid address window ID
IE.ITP   -88.    177650  Invalid TI parameter
IE.IBS   -89.    177647  Invalid send buffer size ( .GT. 255.)
IE.LNL   -90.    177646  LUN locked in use
IE.IUI   -91.    177645  Invalid UIC
IE.IDU   -92.    177644  Invalid device or unit
IE.ITI   -93.    177643  Invalid time parameters
IE.PNS   -94.    177642  Partition/region not in system
IE.IPR   -95.    177641  Invalid priority ( .GT. 250.)
IE.ILU   -96.    177640  Invalid LUN
IE.IEF   -97.    177637  Invalid event flag ( .GT. 64.)
IE.ADP   -98.    177636  Part of DPB out of user's space
IE.SDP   -99.    177635  DIC or DPB size invalid
;
; SUCCESS CODES FROM DIRECTIVES - PLACED IN THE DIRECTIVE STATUS WORD
;
IS.CLR   0       0       EVENT FLAG WAS CLEAR
                                FROM CLEAR EVENT FLAG DIRECTIVE
IS.SET   2       2       EVENT FLAG WAS SET
                                FROM SET EVENT FLAG DIRECTIVE
IS.SPD   2       2       TASK WAS SUSPENDED
```

```
IS.SUP   3       3       LOGICAL NAME SUPERSEDED

;
; THE FOLLOWING LIST IS PROVIDED FOR COMPLETENESS AND INFORMATIVE OR
; SUGGESTIVE PURPOSES.  NOT ALL OF THE I/O FUNCTION CODES AND DEVICES
; ARE SUPPORTED ON P/OS.
;
; COLUMN HEADINGS:
;
;       WORD    CODE   SUBCODE
;    EQUIVALENT (HIGH   (LOW
;               BYTE)   BYTE)
;


;
; GENERAL I/O QUALIFIER BYTE DEFINITIONS
;
IQ.X     000001  000     001    NO ERROR RECOVERY
IQ.Q     000002  000     002    QUEUE REQUEST IN EXPRESS QUEUE
IQ.S     000004  000     004    SYNONYM FOR IQ.UMD
IQ.UMD   000004  000     004    USER MODE DIAGNOSTIC STATUS REQUIRED
IQ.LCK   000200  000     200    MODIFY IMPLIED LOCK FUNCTION
;
; EXPRESS QUEUE COMMANDS
;

IO.KIL   000012  000     012    KILL CURRENT REQUEST
IO.RDN   000022  000     022    I/O RUNDOWN
IO.UNL   000042  000     042    UNLOAD I/O HANDLER TASK
IO.LTK   000050  000     050    LOAD A TASK IMAGE FILE
IO.RTK   000060  000     060    RECORD A TASK IMAGE FILE
IO.SET   000030  000     030    SET CHARACTERISTICS FUNCTION
;
; GENERAL DEVICE HANDLER CODES
;
IO.WLB   000400  001     000    WRITE LOGICAL BLOCK
IO.RLB   001000  002     000    READ LOGICAL BLOCK
IO.LOV   001010  002     010    LOAD OVERLAY (DISK DRIVER)
IO.LDO   001110  002     110    LOAD D-SPACE OVERLAY (DISK)
IO.ATT   001400  003     000    ATTACH A DEVICE TO A TASK
IO.DET   002000  004     000    DETACH A DEVICE FROM A TASK
;
; DIRECTORY PRIMITIVE CODES
;
IO.FNA   004400  011     000    FIND FILE NAME IN DIRECTORY
IO.RNA   005400  013     000    REMOVE FILE NAME FROM DIRECTORY
IO.ENA   006000  014     000    ENTER FILE NAME IN DIRECTORY
;
; FILE PRIMITIVE CODES
;
IO.CLN   003400  007     000    CLOSE OUT LUN
```

QIOSY$

```
IO.ULK   005000  012     000    UNLOCK BLOCK
IO.ACR   006400  015     000    ACCESS FOR READ
IO.ACW   007000  016     000    ACCESS FOR WRITE
IO.ACE   007400  017     000    ACCESS FOR EXTEND
IO.DAC   010000  020     000    DE-ACCESS FILE
IO.RVB   010400  021     000    READ VIRTUAL BLOCK
IO.WVB   011000  022     000    WRITE VIRTUAL BLOCK
IO.EXT   011400  023     000    EXTEND FILE
IO.CRE   012000  024     000    CREATE FILE
IO.DEL   012400  025     000    DELETE FILE
IO.RAT   013000  026     000    READ FILE ATTRIBUTES
IO.WAT   013400  027     000    WRITE FILE ATTRIBUTES
IO.APV   014010  030     010    PRIVILEGED ACP CONTROL
IO.APC   014000  030     000    ACP CONTROL
;
; I/O FUNCTION CODES FOR SPECIFIC DEVICE DEPENDENT FUNCTIONS
;
IO.WLV   000500  001     100    (DECTAPE) WRITE LOGICAL REVERSE
IO.WLS   000410  001     010    (COMM.) WRITE PRECEDED BY SYNC TRAIN
IO.WNS   000420  001     020    (COMM.) WRITE, NO SYNC TRAIN
IO.WAL   000410  001     010    (TTY) WRITE PASSING ALL CHARACTERS
IO.WMS   000420  001     020    (TTY) WRITE SUPPRESSIBLE MESSAGE
IO.CCO   000440  001     040    (TTY) WRITE WITH CANCEL CTRL/O
IO.WBT   000500  001     100    (TTY) WRITE WITH BREAKTHROUGH
IO.WLT   000410  001     010    (DISK) WRITE LAST TRACK
IO.WLC   000420  001     020    (DISK) WRITE LOGICAL W/ WRITECHECK
IO.WPB   000440  001     040    (DISK) WRITE PHYSICAL BLOCK
IO.WDD   000540  001     140    (FLOPPY DISK) WRITE PHYSICAL W/
                                        DELETED DATA
IO.RSN   001140  002     140    (MSCP DISK) READ VOLUME SERIAL NUMBER
IO.RLV   001100  002     100    (MAGTAPE,DECTAPE) READ REVERSE
IO.RST   001001  002     001    (TTY) READ WITH SPECIAL TERMINATOR
IO.RAL   001010  002     010    (TTY) READ PASSING ALL CHARACTERS
IO.RNE   001020  002     020    (TTY) READ WITHOUT ECHO
IO.RNC   001040  002     040    (TTY) READ - NO LOWER CASE CONVERT
IO.RTM   001200  002     200    (TTY) READ WITH TIME OUT
IO.RDB   001200  002     200    (CARD READER) READ BINARY MODE
IO.SCF   001200  002     200    (DISK) SHADOW COPY FUNCTION
IO.RHD   001010  002     010    (COMM.) READ, STRIP SYNC
IO.RNS   001020  002     020    (COMM.) READ, DON'T STRIP SYNC
IO.CRC   001040  002     040    (COMM.) READ, DON'T CLEAR CRC
IO.RPB   001040  002     040    (DISK) READ PHYSICAL BLOCK
IO.RLC   001020  002     020    (DISK,MAGTAPE) READ LOGICAL
                                        W/ READCHEC;**-1
IO.ATA   001410  003     010    (TTY) ATTACH WITH AST'S
IO.GTS   002400  005     000    (TTY) GET TERMINAL SUPPORT
                                        CHARACTERISTICS
IO.R1C   002400  005     000    (AFC,AD01,UDC) READ SINGLE CHANNEL
IO.INL   002400  005     000    (COMM.) INITIALIZATION FUNCTION
IO.TRM   002410  005     010    (COMM.) TERMINATION FUNCTION
IO.RWD   002400  005     000    (MAGTAPE,DECTAPE) REWIND
```

QIOSY$

IO.SPB	002420	005	020	(MAGTAPE) SPACE "N" BLOCKS
IO.RPL	002420	005	020	(DISK) REPLACE LOGICAL BLOCK (RESECTOR)
IO.SPF	002440	005	040	(MAGTAPE) SPACE "N" EOF MARKS
IO.STC	002500	005	100	SET CHARACTERISTIC
IO.SMD	002510	005	110	(FLOPPY DISK) SET MEDIA DENSITY
IO.SEC	002520	005	120	SENSE CHARACTERISTIC
IO.RWU	002540	005	140	(MAGTAPE,DECTAPE) REWIND AND UNLOAD
IO.SMO	002560	005	160	(MAGTAPE) MOUNT & SET CHARACTERISTICS
IO.HNG	003000	006	000	(TTY) HANGUP DIAL-UP LINE
IO.HLD	003100	006	100	(TMS) HANGUP BUT LEAVE LINE ON HOLD
IO.BRK	003200	006	200	(PRO/TTY) SEND SHORT OR LONG BREAK
IO.RBC	003000	006	000	READ MULTICHANNELS (BUFFER DEFINES CHANNELS)
IO.MOD	003000	006	000	(COMM.) SETMODE FUNCTION FAMILY
IO.HDX	003010	006	010	(COMM.) SET UNIT HALF DUPLEX
IO.FDX	003020	006	020	(COMM.) SET UNIT FULL DUPLEX
IO.SYN	003040	006	040	(COMM.) SPECIFY SYNC CHARACTER
IO.EOF	003000	006	000	(MAGTAPE) WRITE EOF
IO.ERS	003020	006	020	(MAGTAPE) ERASE TAPE
IO.DSE	003040	006	040	(MAGTAPE) DATA SECURITY ERASE
IO.RDF	003110	006	110	(DISK) READ DISKETTE FORMAT
IO.RTC	003400	007	000	READ CHANNEL - TIME BASED
IO.SAO	004000	010	000	(UDC) SINGLE CHANNEL ANALOG OUTPUT
IO.SSO	004400	011	000	(UDC) SINGLE SHOT, SINGLE POINT
IO.RPR	004400	011	000	(TTY) READ WITH PROMPT
IO.MSO	005000	012	000	(UDC) SINGLE SHOT, MULTI-POINT
IO.RTT	005001	012	001	(TTY) READ WITH TERMINATOR TABLE
IO.SLO	005400	013	000	(UDC) LATCHING, SINGLE POINT
IO.MLO	006000	014	000	(UDC) LATCHING, MULTI-POINT
IO.LED	012000	024	000	(LPS11) WRITE LED DISPLAY LIGHTS
IO.SDO	012400	025	000	(LPS11) WRITE DIGITAL OUTPUT REGISTER
IO.SDI	013000	026	000	(LPS11) READ DIGITAL INPUT REGISTER
IO.SCS	013000	026	000	(UDC) CONTACT SENSE, SINGLE POINT
IO.REL	013400	027	000	(LPS11) WRITE RELAY
IO.MCS	013400	027	000	(UDC) CONTACT SENSE, MULTI-POINT
IO.ADS	014000	030	000	(LPS11) SYNCHRONOUS A/D SAMPLING
IO.CCI	014000	030	000	(UDC) CONTACT INT - CONNECT
IO.LOD	014000	030	000	(LPA11) LOAD MICROCODE
IO.MDI	014400	031	000	(LPS11) SYNCHRONOUS DIGITAL INPUT
IO.DCI	014400	031	000	(UDC) CONTACT INT - DISCONNECT
IO.PAD	014400	031	000	(PSI) DIRECT CONTROL OF X.29 PAD
HT.RPP	000010	000	010	(PSI) RESET PAD PARAMETERS SUBFUNCTION
IO.XMT	014400	031	000	(COMM.) TRANSMIT SPECIFIED BLOCK WITH ACK
IO.XNA	014410	031	010	(COMM.) TRANSMIT WITHOUT ACK
IO.INI	014400	031	000	(LPA11) INITIALIZE
IO.HIS	015000	032	000	(LPS11) SYNCHRONOUS HISTOGRAM SAMPLING
IO.RCI	015000	032	000	(UDC) CONTACT INT - READ
IO.RCV	015000	032	000	(COMM.) RECEIVE DATA IN BUFFER SPECIFIED

```
IO.CLK   015000  032     000    (LPA11) START CLOCK
IO.CSR   015000  032     000    (BUS SWITCH) READ CSR REGISTER
IO.MDO   015400  033     000    (LPS11) SYNCHRONOUS DIGITAL OUTPUT
IO.CTI   015400  033     000    (UDC) TIMER - CONNECT
IO.CON   015400  033     000    (COMM.) CONNECT FUNCTION
                                (VT11) - CONNECT TASK TO DISPLAY
                                        PROCESSOR
                                (BUS SWITCH) CONNECT TO SPECIFIED BUS
                                (COMM./PRO) DIAL TELEPHONE AND ORIGINATE
IO.ORG   015410  033     010    (COMM.) INITIATE CONNECTION IN
                                        ORIGINATE MODE
IO.ANS   015420  033     020    (COMM.) INITIATE CONNECTION IN
                                ANSWER MODE
IO.STA   015400  033     000    (LPA11) START DATA TRANSFER
                                (XJDRV) - SHOW STATE
IO.DTI   016000  034     000    (UDC) TIMER - DISCONNECT
IO.DIS   016000  034     000    (COMM.) DISCONNECT FUNCTION
                                (VT11) - DISCONNECT TASK FROM DISPLAY
                                        PROCESSOR
                                (BUS SWITCH) SWITCHED BUS DISCONNECT
IO.MDA   016000  034     000    (LPS11) SYNCHRONOUS D/A OUTPUT
IO.DPT   016010  034     010    (BUS SWITCH) DISCONNECT TO SPECIF
                                        PORT NO.
IO.RTI   016400  035     000    (UDC) TIMER - READ
IO.CTL   016400  035     000    (COMM.) NETWORK CONTROL FUNCTION
IO.STP   016400  035     000    (LPS11,LPA11) STOP IN PROGRESS
                                        FUNCTION
                                (VT11) - STOP DISPLAY PROCESSOR
IO.SWI   016400  035     000    (BUS SWITCH) SWITCH BUSSES
IO.CNT   017000  036     000    (VT11) - CONTINUE DISPLAY PROCESSOR
                                (XJDRV) - SHOW COUNTERS
IO.ITI   017000  036     000    (UDC) TIMER - INITIALIZE
;
; PRO 300 SERIES BITMAP FUNCTIONS
;
;        NOTE: THESE FUNCTIONS ARE FOR DEC USE ONLY AND ARE SUBJECT
;        TO CHANGE
;

IO.RSD   006030  014     030    READ SPECIAL DATA
IO.WSD   005410  013     010    WRITE SPECIAL DATA
             SD.TXT    0  TEXT DATA TYPE FOR SPECIAL DATA
             SD.GDS    1  GIDIS DATA TYPE FOR SPECIAL DATA


;
; PROFESSIONAL 300 BISYNC DRIVER (XJDRV) FUNCTIONS
;
SB.PRT   001420  003     020    ATTACH AS A PRINTER
SB.CLR   017010  036     010    CLEAR COUNTERS (IO.CNT
                                SUBFUNCTION)
SB.RDY   015410  033     010    SET DEVICE STATE READY (IO.STA SUBFUNC)
```

```
SB.NRD   015420  033     020     SET DEVICE STATE NOT READY

IO.LBK   016400  035     000     PERFORM LOOPBACK TEST

SB.CBL   016410  035     010     PERFORM CABLE LOOPBACK TEST
SB.CLK   016420  035     020     DEVICE PERFORMS LINE CLOCKING
;
; COMMUNICATIONS FUNCTIONS
;

IO.CPR   015410  033     010     CONNECT NO TIMEOUTS
IO.CAS   015420  033     020     CONNECT WITH AST
IO.CRJ   015440  033     040     CONNECT REJECT
IO.CBO   015510  033     110     BOOT CONNECT
IO.CTR   015610  033     210     TRANSPARENT CONNECT
IO.GNI   016410  035     010     GET NODE INFORMATION
IO.GLI   016420  035     020     GET LINK INFORMATION
IO.GLC   016430  035     030     GET LINK INFO CLEAR COUNTERS
IO.GRI   016440  035     040     GET REMOTE NODE INFORMATION
IO.GRC   016450  035     050     GET REMOTE NODE ERROR COUNTS
IO.GRN   016460  035     060     GET REMOTE NODE NAME
IO.CSM   016470  035     070     CHANGE SOLO MODE
IO.CIN   016500  035     100     CHANGE CONNECTION INHIBIT
IO.SPW   016510  035     110     SPECIFY NETWORK PASSWORD
IO.CPW   016520  035     120     CHECK NETWORK PASSWORD.
IO.NLB   016530  035     130     NSP LOOPBACK
IO.DLB   016540  035     140     DDCMP LOOPBACK

;
; ICS/ICR I/O FUNCTIONS
;
IO.CTY   003400  007     000     CONNECT TO TERMINAL INTERRUPTS
IO.DTY   006400  015     000     DISCONNECT FROM TERMINAL INTERRUPTS
IO.LDI   007000  016     000     LINK TO DIGITAL INTERRUPTS
IO.UDI   011410  023     010     UNLINK FROM DIGITAL INTERRUPTS
IO.LTI   007400  017     000     LINK TO COUNTER MODULE INTERRUPTS
IO.UTI   011420  023     020     UNLINK FROM COUNTER MODULE INTERRUPTS
IO.LTY   010000  020     000     LINK TO REMOTE TERMINAL INTERRUPTS
IO.UTY   011430  023     030     UNLINK FROM REMOTE TERMINAL INTERRUPTS
IO.LKE   012000  024     000     LINK TO ERROR INTERRUPTS
IO.UER   011440  023     040     UNLINK FROM ERROR INTERRUPTS
IO.NLK   011400  023     000     UNLINK FROM ALL INTERRUPTS
IO.ONL   017400  037     000     UNIT ONLINE
IO.FLN   012400  025     000     UNIT OFFLINE
IO.RAD   010400  021     000     READ ACTIVATING DATA

;
; IP11 I/O FUNCTIONS
;
IO.MAO   003410  007     010     MULTIPLE ANALOG OUTPUTS
IO.LEI   007410  017     010     LINK EVENT FLAGS TO INTERRUPT
```

```
IO.RDD   010010  020     010    READ DIGITAL DATA
IO.RMT   010020  020     020    READ MAPPING TABLE
IO.LSI   011000  022     000    LINK TO DSI INTERRUPTS
IO.UEI   011450  023     050    UNLINK EVENT FLAGS
IO.USI   011460  023     060    UNLINK FROM DSI INTERRUPTS
IO.CSI   013000  026     000    CONNECT TO DSI INTERRUPTS
IO.DSI   013400  027     000    DISCONNECT FROM DSI INTERRUPTS
IO.RAM   015000  032     000    READ ANALOG MAPPING TABLES

;
; PCL11 I/O FUNCTIONS
;

IO.ATX   000400  001     000    ATTEMPT TRANSMISSION
IO.ATF   001000  002     000    ACCEPT TRANSFER
IO.CRX   014400  031     000    CONNECT FOR RECEPTION
IO.DRX   015000  032     000    DISCONNECT FROM RECEPTION
IO.RTF   015400  033     000    REJECT TRANSFER

;
; DEFINE THE GENERAL USER-MODE I/O QUALIFIER BIT.
;

IQ.UMD   000004  000     004    USER MODE DIAGNOSTIC REQUEST

;
; DEFINE USER-MODE DIAGNOSTIC FUNCTIONS.
;

IO.HMS   004000  010     000    (DISK) HOME SEEK OR RECALIBRATE
IO.BLS   004010  010     010    (DISK) BLOCK SEEK
IO.OFF   004020  010     020    (DISK) OFFSET POSITION
IO.RDH   004030  010     030    (DISK) READ DISK HEADER
IO.WDH   004040  010     040    (DISK) WRITE DISK HEADER
IO.WCK   004050  010     050    (DISK) WRITECHECK (NON-TRANSFER)
IO.RNF   004060  010     060    (DECTAPE) READ BLOCK NUMBER FORWARD
IO.RNR   004070  010     070    (DECTAPE) READ BLOCK NUMBER REVERSE
IO.RDS   004070  010     070    (DISK-RX50) RETURN DISKETTE SPEED
IO.LPC   004100  010     100    (MAGTAPE) READ LONGITUDINAL PARITY
                                        CHAR
IO.RTD   004120  010     120    (DISK) READ TRACK DESCRIPTOR
IO.WTD   004130  010     130    (DISK) WRITE TRACK DESCRIPTOR
IO.TDD   004140  010     140    (DISK) WRITE TRACK DESCRIPTOR
                                        DISPLACED
IO.DGN   004150  010     150    DIAGNOSE MICRO PROCESSOR FIRMWARE
IO.WPD   004160  010     160    (DISK) WRITE PHYSICAL BLOCK
IO.RPD   004170  010     170    (DISK) READ PHYSICAL BLOCK
IO.CER   004200  010     200    (DISK) READ CE BLOCK
IO.CEW   004210  010     210    (DISK) WRITE CE BLOCK
```

A.21 SCBDF$

```
;+
; STATUS CONTROL BLOCK
;
; THE STATUS CONTROL BLOCK (SCB) DEFINES THE STATUS OF A DEVICE
; CONTROLLER. THERE IS ONE SCB FOR EACH CONTROLLER IN A SYSTEM.
; THE SCB IS POINTED TO BY UNIT CONTROL BLOCKS. NORMALLY, AN
; SCB EXISTS FOR EACH "PROCESS" THAT A DRIVER CAN POSSIBLY HAVE
; CONCURRENTLY ACTIVE. FOR EXAMPLE, A DISK THAT SUPPORTS
; OVERLAPPED SEEK OPERATIONS REQUIRES THAT THERE EXIST A
; SEPARATE SCB FOR EACH UNIT THAT CAN BE ACTIVE. IN THIS CASE,
; THE FORKBLOCK IN THE SCB IS USED TO SUSPEND THE DRIVER PROCESS
; DURING GTPKT AND IS LINKED INTO THE CONTROLLER REQUEST QUEUE
; WHOSE LISTHEAD IS IN THE KRB OF THE ASSIGNED CONTROLLER
; (USUALLY STATIC S.KRB).
;
; IN ADDITION TO CONTAINING THE DRIVER'S PROCESS CONTEXT AND
; DEFINING THE STATE OF THE CURRENT CONTROLLER STATE (NOT
; NECESSARILY THE PHYSICAL CONTROLLER STATE) A LISTHEAD IS
; MAINTAINED OF ALL PENDING I/O PACKETS TO BE PROCESSED.
;
;-
.IF NB   SYSDF

         .ASECT
.=0
000000   S.LHD:  .BLKW   2       ;PENDING I/O REQUEST (PACKET) QUEUE
                                 ;       LISTHEAD
000004   S.FRK:  .BLKW   1       ;FORK BLOCK LINK WORD
000006           .BLKW   1       ;FORK-PC
000010           .BLKW   1       ;FORK-R5
000012           .BLKW   1       ;FORK-R4
000014   S.KS5:  .BLKW   1       ;FORK KISAR5
000016   S.PKT:  .BLKW   1       ;ADDRESS OF CURRENT I/O PACKET
000020   S.CTM:  .BLKB   1       ;CURRENT TIMEOUT COUNT
000021   S.ITM:  .BLKB   1       ;INITIAL TIMEOUT COUNT
000022   S.STS:  .BLKB   1       ;STATUS (0=FREE, NE 0=BUSY)
000023   S.ST3:  .BLKB   1       ;STATUS EXTENSION BYTE
000024   S.ST2:  .BLKW   1       ;STATUS EXTENSION
000026   S.KRB:  .BLKW   1       ;ADDRESS OF KRB
000030   S.RCNT: .BLKB   1       ;NUMBER OF REGISTERS TO COPY
                                 ;(NOT IN USE )
000031   S.ROFF: .BLKB   1       ;OFFSET TO FIRST DEV REG TO COPY
                                 ;(NOT IN USE)
000032   S.EMB:  .BLKW   1       ;ERROR MESSAGE BLOCK POINTER
                                 ;(NOT IN USE)
000034   S.KTB:  .BLKW   1       ;START OF MULTI-ACCESS KRBS (OPTIONAL)

         .PSECT
```

SCBDF$

```
        .IFF

;+
; STATUS CONTROL BLOCK STATUS EXTENSION BIT DEFINITIONS
;-

        S2.EIP=1                ;ERROR IN PROGRESS (1=YES)
        S2.ENB=2                ;ERROR LOGGING ENABLED (0=YES)
        S2.LOG=4                ;ERROR LOGGING SUPPORTED (1=YES)
        S2.MAD=10               ;MULTIACCESS DEVICE (1=YES)
        S2.LDS=40               ;LOAD SHARING ENABL. (1=YES, MUST BE
                                ;0 ON P/OS)
        S2.OPT=100              ;SUPPORTS EXECUTIVE BLOCK CHECKING
                                ;(MAX LBN)
                                ;SUPPORTS ACCOUNTING STATISTICS
                                ;AND MIGHT SUPPORT SEEK OPTIMIZATION
                                ;(DEPENDS ON S3.OPT)
        S2.CON=200              ;SCB AND KRB ARE CONTIGUOUS (1=YES)
        S2.OP1=400              ;THESE TWO BITS DEFINE THE OPTIMIZATION
        S2.OP2=1000             ;METHOD.
                                ;OP2,OP1=0,0 INDICATES NEAREST CYLINDER
                                ;OP2,OP1=0,1 INDICATES ELEVATOR
                                ;OP2,OP1=1,0 INDICATES C-SCAN
                                ;OP2,OP1=1,1 RESERVED
        S2.ACT=2000             ;DRIVER HAS OPERATION OUTSTANDING
                                ;(1=YES)
        S2.XHR=4000             ;EXTERNAL HEADER AND NEW I.LN2 SUPPORT
                                ;(0 FOR FILES-11 OR ANSI MAGTAPE ACPS)


;+
; STATUS CONTROL BLOCK STATUS EXTENSION (S.ST3) DEFINITIONS
;-
        S3.DRL=1                ;MULTI-ACCESS DRIVE IN RELEASED STATE
                                ;(1=YES)
        S3.NRL=2                ;DRIVER SHOULDN'T RLS MULTI-ACCESS
                                ;DRIVE (1=YES)
        S3.SIP=4                ;SEEK IN PROGRESS (1=YES)
        S3.ATN=10               ;DRIVER MUST CLEAR ATTENTION BIT
                                ;(1=YES)
        S3.SLV=20               ;DEVICE USES SLAVE UNITS (1=YES)
        S3.SPA=40               ;PORT 'A' SPINNING UP
        S3.SPB=100              ;PORT  SPINNING UP
        S3.OPT=200              ;SEEK OPTIMIZATIONS ENABLED (1=YES)
        S3.SPU=S3.SPA!S3.SPB    ;.OR. OF PORT SPINUP BITS

;+
; KRB ADDRESS TABLE (S.KTB) PORT OFFLINE FROM THIS SCB FLAG.
;-
        KP.OFL=1        ;KRB ADDRESS POINTS TO OFFLINE PORT (1=YES)
```

SCBDF$

```
;+
; MAPPING ASSIGNMENT BLOCK (FOR UNIBUS MAPPING REGISTER
; ASSIGNMENT) ..OF HISTORICAL INTEREST ONLY ON P/OS AND
; QBUS PROCESSORS
;-
                .ASECT
        .=0
        M.LNK:  .BLKW   1       ;LINK WORD
        M.UMRA: .BLKW   1       ;ADDRESS OF FIRST ASSIGNED UMR
        M.UMRN: .BLKW   1       ;NUMBER OF UMR'S ASSIGNED * 4
        M.UMVL: .BLKW   1       ;LOW 16 BITS MAPPED BY 1ST ASSIGNED
                                ;UMR
        M.UMVH: .BLKB   1       ;HIGH 2 BITS MAPPED IN BITS 4 AND 5
        M.BFVH: .BLKB   1       ;HIGH 6 BITS OF PHYSICAL BUFFER
                                ;ADDRESS
        M.BFVL: .BLKW   1       ;LOW 16 BITS OF PHYSICAL BUFFER
                                ;ADDRESS
        M.LGTH= .               ;LENGTH OF MAPPING ASSIGNMENT BLOCK

        .ENDC

        .PSECT
```

A.22 TTSYM$

	DECIMAL	OCTAL	
TC.WID	1.	1	LINE WIDTH
TC.LPP	2.	2	LINES PER PAGE
TC.RSP	3.	3	RECEIVER SPEED
TC.XSP	4.	4	TRANSMITTER SPEED
TC.STB	5.	5	TWO STOP BITS
TC.ISL	6.	6	SUBLINE ON INTERFACE
TC.RAT	7.	7	READ-AHEAD TYPE
TC.TTP	8.	10	TERMINAL TYPE
TC.SCR	9.	11	SCRIPT LINE
TC.SCP	10.	12	SCOPE
TC.HFL	11.	13	HORIZONTAL FILL REQUIREMENT
TC.VFL	12.	14	VERTICAL FILL
TC.NL	13.	15	ASCII NEWLINE TERMINAL
TC.SFF	14.	16	SIMULATE FORMFEED AND VERTAB
TC.HFF	15.	17	HARDWARE FORMFEED AND VERTAB
TC.LVF	16.	20	LA36 VERTICAL FILL
TC.HHT	17.	21	HARDWARE HORIZONTAL TAB
TC.NST	18.	22	NON-STANDARD HARDWARE TAB
TC.BSP	19.	23	HARDWARE BACKSPACE
TC.ACR	20.	24	AUTOMATIC CARRIAGE RETURN REQUIRED
TC.SMR	21.	25	SMALL CHARACTER INPUT ENABLED
TC.SMP	22.	26	SMALL CHARACTER INPUT REQUIRED (/LOWERCASEINPUT)
TC.SMO	23.	27	SMALL CHARACTER OUTPUT ENABLED
TC.CCF	24.	30	CTRL/C FLUSHES TYPEAHEAD AND READ
TC.ALT	25.	31	ALTERNATIVE ALTMODE RECOGNITION
TC.IMG	26.	32	IAS - MESSAGES INHIBITED
TC.NKB	27.	33	NO KEYBOARD
TC.NPR	28.	34	NO PRINTER
TC.ESQ	29.	35	ESCAPE SEQUENCE RECOGNITION
TC.LCP	30.	36	LOCAL COPY LINE
TC.PAR	31.	37	PARITY RECOGNITION/GENERATION REQUIRED
TC.EPA	32.	40	EVEN PARITY
TC.DLU	33.	41	DIALUP LINE
TC.BLK	34.	42	BLOCK MODE TERMINAL
TC.FRM	35.	43	FORMS MODE TERMINAL
TC.HLD	36.	44	TERMINAL HOLD MODE
TC.TAP	37.	45	LOW SPEED PAPER TAPE READER
TC.CEQ	38.	46	COMPATIBLE ESCAPE SEQUENCES
TC.NEC	39.	47	TERMINAL IN NO-ECHO MODE
TC.SLV	40.	50	TERMINAL IN SLAVE MODE
TC.PRI	41.	51	TERMINAL IS PRIVILEGED
TC.UC0	42.	52	USER CHARACTERISTIC 0
TC.UC1	43.	53	USER CHARACTERISTIC 1
TC.UC2	44.	54	USER CHARACTERISTIC 2
TC.UC3	45.	55	USER CHARACTERISTIC 3
TC.UC4	46.	56	USER CHARACTERISTIC 4

TTSYM$

TC.UC5	47.	57	USER CHARACTERISTIC 5
TC.UC6	48.	60	USER CHARACTERISTIC 6
TC.UC7	49.	61	USER CHARACTERISTIC 7
TC.UC8	50.	62	USER CHARACTERISTIC 8
TC.UC9	51.	63	USER CHARACTERISTIC 9
TC.FDX	52.	64	LINE HAS FULL DUPLEX CAPABILITY
TC.BIN	53.	65	TERMINAL HAS BINARY INPUT (DO NOT RECOGNIZE IMMEDIATE CONTROL CHARS)
TC.REM	54.	66	TERMINAL IS REMOTE (CONNECTED VIA MODEM)
TC.8BC	55.	67	ACCEPT 8-BIT CHARACTERS
TC.P8B	56.	70	PASS 8 BITS ON READ PASS ALL
TC.TBF	57.	71	TYPEAHEAD BUFFER CHARACTER COUNT
TC.CTS	58.	72	CTRL/S STATE
TC.ANS	59.	73	ESCAPE SEQUENCES ARE ANSI FORMAT
TC.CSQ	60.	74	ONLY PASS CTRL/S,Q ON READ PASS ALL
TC.CTC	61.	75	ONLY PASS CTRL/S,Q,C ON REAL PASS ALL
TC.ASP	62.	76	REMOTE ANSWER SPEED
TC.ABD	63.	77	AUTO-BAUD SPEED DETECTION
TC.TBS	64.	100	TYPEAHEAD BUFFER SIZE
TC.TBM	65.	101	TYPEAHEAD BUFFER MODE (TASK OR CLI)
TC.NBR	66.	102	DON'T BROADCAST TO THIS TERMINAL
TC.ACD	67.	103	ANCILLARY CONTROL DRIVER
TC.ARC	68.	104	AUTOANSWER RING COUNT
TC.TRN	69.	105	SET DIALING TRANSLATE TABLE
TC.XMM	70.	106	MAINTENANCE MODE
TC.FSZ	71.	107	FRAME SIZE, DATA BITS + PARITY BIT (IF ANY)
XT.DLM	72.	110	DIALING MODE (TONE, PULSE, ...)
XT.DMD	73.	111	DATA MODE (SERIAL, CODEC, VOICE, DTMF)
XT.DTT	74.	112	DTMF TONE LENGTH (10MS MULTIPLES)
XT.DIT	75.	113	DTMF INTERDIGIT LENGTH
XT.MTP	76.	114	MODEM TYPE
XT.SDE	77.	115	SET DTMF ESCAPE SEQUENCE
XT.TAK	78.	116	AUX KEYBOARD INTERPRETATION EN/DISABLE
XT.GOV	79.	117	GO VOICE (I.E. DELAY LOSS OF CARRIER DISCONNECT)
XT.TSP	80.	120	MONITOR LINE (TURN TMS SPEAKER ON/OFF)
XT.TTO	81.	121	HARDWARE SILENCE TIMEOUT (SECONDS)
TC.ANI	82.	122	ANSI CRT TERMINAL
TC.AVO	83.	123	VT100-FAMILY TERMINAL DISPLAY
TC.DEC	84.	124	DIGITAL CRT TERMINAL
TC.EDT	85.	125	LOCAL TERMINAL EDITING
TC.RGS	86.	126	REGIS GRAPHICS
TC.INT	87.	127	HANDLE ^C (OR INT-DO) DIFFERENTLY FOR IO.RTT FOR P/OS
TC.TLC	88.	130	INDICATE CLI SHOULD GET ^C NOTIFICATION
TC.MAX	89.	131	THIS MUST BE ONE GREATER THAN THE HIGHEST VALUE USED FOR A SYMBOL

```
;+
; SET CHARACTERISTIC ERROR CODES
;-
```

```
SE.ICN   1.      1        ILLEGAL CHARACTERISTIC NAME
SE.FIX   2.      2        ATTEMPT TO CHANGE FIXED CHARACTERISTIC
SE.BIN   3.      3        ILLEGAL VALUE FOR BINARY CHARACTERISTIC
SE.VAL   4.      4        ILLEGAL VALUE FOR NON-BINARY CHARACTERISTIC
SE.TER   5.      5        ILLEGAL TERMINAL TYPE
SE.SPD   6.      6        ILLEGAL SPEED FOR INTERFACE
SE.SPL   7.      7        ILLEGAL SPLIT SPEED FOR INTERFACE
SE.PAR   8.      10       ILLEGAL PARITY TYPE FOR INTERFACE
SE.LPR   9.      11       OTHER ILLEGAL LINE PARAMETERS
SE.NSC   10.     12       INTERFACE DOES NOT HAVE SETTABLE
                          CHARACTERISTICS
SE.UPN   11.     13       NO SPACE TO SAVE DEFAULT CHARACTERISTICS
SE.NIH   12.     14       CHARACTERISTIC NOT ASSEMBLED IN HANDLER

;+
; NOW THE SUBFUNCTION CODES FOR THE SET CHARACTERISTICS FUNCTION
;-

SF.SSC           2400!020       SET SINGLE CHARACTERISTIC
SF.SMC           2400!040       SET MULTIPLE CHARACTERISTICS
SF.RDF           2400!060       RESTORE DEFAULT
SF.STT           2400!100       SET TERMINAL TYPE
SF.STS           2400!120       SET TERMINAL TYPE AND SPEED
SF.GSC           2400!140       GET SINGLE CHARACTERISTIC
SF.GMC           2400!160       GET MULTIPLE CHARACTERISTICS
SF.GAC           2400!200       GET ALL CHARACTERISTICS
SF.SAC           2400!220       SET ALL CHARACTERISTICS

SF.DEF           010            SET DEFAULT CHARACTERISTICS

;+
; NOW THE SPEED TYPES
;-

S.0      1       1.
S.50     2       2.
S.75     3       3.
S.100    4       4.
S.110    5       5.
S.134    6       6.
S.150    7       7.
S.200    8       10.
S.300    9       11.
S.600    10      12.
S.1200   11      13.
S.1800   12      14.
S.2000   13      15.
S.2400   14      16.
S.3600   15      17.
S.4800   16      20.
```

```
S.7200   17       21.
S.9600   18       22.
S.EXTA   19       23.
S.EXTB   20       24.
S.19.2   21.      25          19.2KBPS
S.38.4   22.      26          38.4KBPS

;+
; NOW THE TERMINAL TYPES
;-

T.UNK0   0.       0        UNKNOWN (UNSPECIFIED)
T.AS33   1.       1        ASR33
T.KS33   2.       2        KSR33
T.AS35   3.       3        ASR35
T.L30S   4.       4        LA30S
T.L30P   5.       5        LA30P
T.LA36   6.       6        LA36
T.VT05   7.       7        VT05
T.VT50   8.       10       VT50
T.VT52   9.       11       VT52
T.VT55   10.      12       VT55
T.VT61   11.      13       VT61
T.L180   12.      14       LA180S
T.V100   13.      15       VT100
T.L120   14.      16       LA120
T.SCR0   15.      17       SCRIPT LINE
T.LA12   16.      20       LA12
T.L100   17.      21       LA100
T.LA34   18.      22       LA34
T.LA38   19.      23       LA38
T.V101   20.      24       VT101
T.V102   21.      25       VT102
T.V105   22.      26       VT105
T.V125   23.      27       VT125
T.V131   24.      30       VT131
T.V132   25.      31       VT132
T.LA50   26.      32       LA50
T.LQP1   27.      33       LQP I
T.LQP2   28.      34       LQP II
T.BMP1   29.      35       BIT MAP TERMINAL #1 (ORIGINAL PRO 350)
T.USR0   128.     200      USER TERMINAL 0
T.USR1   T.USR0+1          USER TERMINAL 1
T.USR2   T.USR1+1          USER TERMINAL 2
T.USR3   T.USR2+1          USER TERMINAL 3
T.USR4   T.USR3+1          USER TERMINAL 4
 ;
; DIAL MODES

XT.DIA   0        0        DIAL PULSE, 10 PULSES PER SEC
XT.DTM   1        1        DTMF
```

```
XT.D20   2       2       DIAL PULSE, 20 PULSES PER SEC
XT.OHS   3       3       OFF-HOOK SERVICE (EXTERNAL FIXED NUMBER)

; DATA MODES

XT.VOI   0       0       VOICE TELEPHONE (NO DATA)
XT.SER   1       1       SERIAL DATA
XT.ENC   2       2       ENCODED VOICE
XT.DTD   3       3       DTMF DATA

; MODEM TYPE

XTM.NO   -1.     177777  NO MODEM, HARD-WIRED LINE
XTM.FS   0       0       US FSK, 0..300 BAUD BELL 103
XTM.PS   1       1       US DPSK, 1200 BAUD BELL 212
XTM.21   5       5       CCITT V.21, 0..300 BAUD EUROPEAN
XTM.M1   6       6       CCITT V.23 MODE 1, 75/0..600
XTM.M2   7       7       CCITT V.23 MODE 2, 75/0..1200
XTM.US   10.     12      MICRO-SWITCH

; UNSOLICITED EVENT TYPES

XTU.UI   0       0       UNSOLICITED INPUT
XTU.CD   2       2       CARRIER DETECT NOTIFICATION
XTU.CL   4       4       ARRIER LOSS
XTU.DR   6       6       DTMF ESCAPE SEQUENCE RECEIVED
XTU.OF   8.      10      XOFF RECEIVED
XTU.ON   10.     12      XON RECEIVED
XTU.RI   12.     14      RING
XTU.TU   14.     16      TELSET OFF HOOK
XTU.TD   16.     20      TELSET ON HOOK


;
; BITS FOR RETURN FROM 'GET TERMINAL SUPPORT'
;
       BIT MASK

F1.ACR   000001          AUTO CR/LF ON LONG LINES
F1.BTW   000002          BREAK THROUGH WRITE
F1.BUF   000004          INTERMEDIATE BUFFERING
F1.UIA   000010          UNSOLICITED INPUT AST'S
F1.CCO   000020          CANCEL CTRL-O ON WRITE
F1.ESQ   000040          ESCAPE SEQUENCE SUPPORT
F1.HLD   000100          HOLD SCREEN SUPPORT
F1.LWC   000200          LOWER CASE CONVERSION
F1.RNE   000400          READ NO ECHO
F1.RPR   001000          READ WITH PROMPT
F1.RST   002000          READ WITH SPECIAL TERMINATORS
F1.RUB   004000          SCOPE RUBOUTS
F1.SYN   010000          XON/XOFF
```

TTSYM$

```
F1.TRW   020000          TRANSPARENT READ/WRITE
F1.UTB   040000          BUFFERING IN TASK BUFFER
F1.VBF   100000          EXEC BUFFERS ARE VARIABLE LENGTH
F2.SCH   000001          SET CHARACTERISTICS
F2.GCH   000002          GET CHARACTERISTICS
F2.DCH   000004          DUMP/RESTORE CHARACTERISTICS
F2.DKL   000010          HISTORICAL 11D/IAS IO.KIL
F2.ALT   000020          ALTMODE IS ECHOED
F2.SFF   000040          FORMFEED CAN BE SIMULATED
F2.CUP   000100          DEVICE INDEPENDENT CURSOR POSITIONING
F2.FDX   000200          FULL DUPLEX TERMINAL DRIVER

;
; SUBFUNCTION BITS FOR TERMINAL HANDLER QIO'S.  NOTE THAT THIS
; MUST REFLECT THE REALITY IN QIOMAC.MAC.
;

TF.RST   001             [IO.RLB/IO.RPR] READ WITH SPECIAL TERMINATORS
TF.BIN   002                    SEND PROMPT AS 'PASS ALL'
TF.RAL   010                    READ PASS ALL
TF.RNE   020                    READ WITH NO ECHO
TF.RNC   040                    READ WITH NO CASE CONVERSION
TF.XOF   100                    SEND XOF AFTER PROMPT
TF.TMO   200                    READ WITH TIMEOUT
TF.RCU   001             [IO.WLB] RESTORE CURSOR POSITION
TF.WAL   010                    WRITE PASS ALL
TF.WMS   020                    WRITE SUPPRESSIBLE MESSAGE
TF.CCO   040                    CANCEL CTRL/O
TF.WBT   100                    BREAK THROUGH READ
TF.SYN   200                    SYNCHRONOUS WRITE (IAS ONLY)
TF.XCC   001              [IO.ATT] DO NOT TRAP CTRL/C
TF.NOT   002                    NOTIFICATION ONLY FOR TYPEAHEAD
TF.AST   010                    SPECIFY AST'S IN ATTACH
TF.ESQ   020                    RECOGNIZE ESCAPE SEQUENCES
TF.UCH   040                    CHARACTER AST NOTIFICATION (IAS)
```

A.23 TCBDF$

```
;+
; TASK CONTROL BLOCK OFFSET AND STATUS DEFINITIONS
;
; TASK CONTROL BLOCK
;-
                  .ASECT
         .=0
000000   T.LNK:   .BLKW   1       ;UTILITY LINK WORD
000002   T.PRI:   .BLKB   1       ;TASK PRIORITY
000003   T.IOC:   .BLKB   1       ;I/O PENDING COUNT
000004   T.PCBV:  .BLKW   1       ;POINTER TO COMMON PCB VECTOR
000006   T.NAM:   .BLKW   2       ;TASK NAME IN RADIX-50
000012   T.RCVL:  .BLKW   2       ;RECEIVE QUEUE LISTHEAD
000016   T.ASTL:  .BLKW   2       ;AST QUEUE LISTHEAD
000022   T.EFLG:  .BLKW   2       ;TASK LOCAL EVENT FLAGS 1-32
000026   T.UCB:   .BLKW   1       ;UCB ADDRESS FOR PSEUDO DEVICE 'TI'
000030   T.TCBL:  .BLKW   1       ;TASK LIST THREAD WORD
000032   T.STAT:  .BLKW   1       ;FIRST STATUS WORD (BLOCKING BITS)
000034   T.ST2:   .BLKW   1       ;SECOND STATUS WORD (STATE BITS)
000036   T.ST3:   .BLKW   1       ;THIRD STATUS WORD (ATTRIBUTE BITS)
000040   T.DPRI:  .BLKB   1       ;TASK'S DEFAULT PRIORITY
000041   T.LBN:   .BLKB   3       ;LBN OF TASK LOAD IMAGE
000044   T.LDV:   .BLKW   1       ;UCB ADDRESS OF LOAD DEVICE
000046   T.PCB:   .BLKW   1       ;PCB ADDRESS OF TASK PARTITION
000050   T.MXSZ:  .BLKW   1       ;MAXIMUM SIZE OF TASK IMAGE (MAPPED
                                  ;ONLY)
000052   T.ACTL:  .BLKW   1       ;ADDRESS OF NEXT TASK IN ACTIVE LIST
000054   T.ATT:   .BLKW   2       ;ATTACHMENT DESCRIPTOR LISTHEAD
000060   T.ST4:   .BLKW   1       ;FOURTH TASK STATUS WORD
000062   T.HDLN:  .BLKB   1       ;LENGTH OF HEADER (0 IF HDR IN POOL)
000063            .BLKB   1       ;UNUSED
000064   T.GGF:   .BLKB   1       ;GROUP GLOBAL USE COUNT FOR TASK
000065   T.TIO:   .BLKB   1       ;BUFFERED I/O IN PROGRESS COUNT
000066   T.EFLM:  .BLKW   2       ;TASK WAITFOR MASK/ADDRESS
000072   T.TKSZ:  .BLKW   1       ;TASK LOAD SIZE IN 32 WD BLOCKS
000074   T.OFF:   .BLKW   1       ;OFFSET TO TASK IMAGE IN PARTITION
000076            .BLKB   1       ;RESERVED
000077   T.SRCT:  .BLKB   1       ;SREF WITH EFN COUNT IN ALL RECEIVE
                                  ;QUEUES
000100   T.RRFL:  .BLKW   2       ;RECEIVE BY REFERENCE LISTHEAD
000104   T.OCBH:  .BLKW   2       ;OFFSPRING CONTROL BLOCK LISTHEAD
000110   T.RDCT:  .BLKW   1       ;OUTSTANDING OFFSPRING AND VT: COUNT
000112   T.SAST:  .BLKW   1       ;SPECIFY AST LISTHEAD

                  .IF NB  SYSDF
         $$$ = .

000114   T.RRM:   .BLKW   1       ;REQUIRED RUN MASK (NOT USED)
000116   T.IRM:   .BLKW   1       ;INITIAL RUN MASK SET UP BY INSTALL
```

```
                                       ;(NOT USED)
000120   T.CPU:  .BLKB   1       ;PROCESSOR NUMBER ON WHICH TASK LAST
                                       ;EXECUTED
000121           .BLKB   1       ;(UNUSED)

         .=$$$

         $$$=.

         T.ACN:  .BLKW 1 ;POINTER TO ACCOUNTING BLOCK


                 .IF NDF A$$CNT ;NOT CURRENTLY USED

         .=$$$

                 .ENDC

         $$$=.

         T.ISIZ: .BLKW 1 ;SIZE OF ROOT I SPACE

                 .IF NDF U$$DAS

         .=$$$

                 .ENDC           ; NDF U$$DAS


         T.LGTH=.                ;LENGTH OF TASK CONTROL BLOCK
         T.EXT=0                 ;LENGTH OF TCB EXTENSION

                 .IFF


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

         TS.EXE=100000   ;TASK NOT IN EXECUTION (1=YES)
         TS.RDN=40000    ;I/O RUN DOWN IN PROGRESS (1=YES)
         TS.MSG=20000    ;ABORT MESSAGE BEING OUTPUT (1=YES)
         TS.CIP=10000    ;TASK BLOCKED FOR CHECKPOINT IN PROGRESS
                         ;(1=YES)
         TS.RUN=4000     ;TASK IS RUNNING ON ANOTHER PROCESSOR (1=YES)
         TS.STP=1000     ;TASK BLOCKED BY CLI COMMAND
         TS.CKR=100      ;TASK HAS CKP REQUEST (MP SYSTEM ONLY) (1=YES)
         TS.BLC=37       ;INCREMENT BLOCKING COUNT MASK
```

TCBDF$

```
;+
; TASK BLOCKING STATUS MASK
;-

        TS.BLK=177777

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

        T2.AST=100000   ;AST IN PROGRESS (1=YES)
        T2.DST=40000    ;AST RECOGNITION DISABLED (1=YES)
        T2.CHK=20000    ;TASK NOT CHECKPOINTABLE(1=YES,SEE PCB AS
                        ;WELL)
        T2.REX=10000    ;REQUESTED EXIT AST SPECIFIED
        T2.SEF=4000     ;TASK STOPPED FOR EVENT FLAG(S) (1=YES)
        T2.SIO=1000     ;TASK STOPPED FOR BUFFERED I/O
        T2.AFF=400      ;TASK IS INSTALLED WITH AFFINITY
        T2.HLT=200      ;TASK IS BEING HALTED (1=YES)
        T2.ABO=100      ;TASK MARKED FOR ABORT (1=YES)
        T2.STP=40       ;SAVED T2.SPN ON AST IN PROGRESS
        T2.STP=20       ;TASK STOPPED (1=YES)
        T2.SPN=10       ;SAVED T2.SPN ON AST IN PROGRESS
        T2.SPN=4        ;TASK SUSPENDED (1=YES)
        T2.WFR=2        ;SAVED T2.WFR ON AST IN PROGRESS
        T2.WFR=1        ;TASK IN WAITFOR STATE (1=YES)
;+
; THIRD STATUS WORD (ATTRIBUTE BITS)
;-

        T3.ACP=100000   ;ANCILLARY CONTROL PROCESSOR (1=YES)
        T3.PMD=40000    ;DUMP TASK ON SYNCHRONOUS ABORT (0=YES)
        T3.REM=20000    ;REMOVE TASK ON EXIT (1=YES)
        T3.PRV=10000    ;TASK IS PRIVILEGED (1=YES)
        T3.MCR=4000     ;TASK REQUESTED AS EXTERNAL MCR FUNCTION
                        ;(1=YES)
        T3.SLV=2000     ;TASK IS A SLAVE TASK (1=YES)
        T3.CLI=1000     ;TASK IS A COMMAND LINE INTERPRETER (1=YES)
        T3.RST=400      ;TASK IS RESTRICTED (1=YES)
        T3.NSD=200      ;TASK DOES NOT ALLOW SEND DATA
        T3.CAL=100      ;TASK HAS CHECKPOINT SPACE IN TASK IMAGE
        T3.ROV=40       ;TASK HAS RESIDENT OVERLAYS
        T3.NET=20       ;NETWORK PROTOCOL LEVEL
        T3.MPC=10       ;MAPPING CHANGE WITH OUTSTANDING I/O (1=YES)
        T3.CMD=4        ;TASK IS EXECUTING A CLI COMMAND
        T3.SWS=2        ;RESERVED FOR SPM-11, (NOT AVAILABLE ON P/OS,
                        ;THIS BIT HAS BEEN TEMPORARILY REDEFINED
                        ;TO DISTINGUISH APPLICATION TASKS
                        ;FROM SYSTEM TASKS. THE FORMER ARE REMOVED
                        ;WHEN APPLICATION EXITS. IT IS SOMETIMES
                        ;REFERRED TO AS THE "BEAT ME" BIT).
```

```
         T3.GFL=1        ;GROUP GLOBAL EVENT FLAG LOCK

;+
; STATUS BIT DEFINITIONS FOR FOURTH STATUS WORD (T.ST4)
;-

         T4.VCT=100      ;TASK HAS BEEN VICTIMIZED (BLASTED)
         T4.MUT=40       ;TASK IS A MULTI-USER TASK (MEANING SEPARATED
                         ;READ ONLY AND READ/WRITE FOR TASK REGION. THIS
                         ;HAS PERFORMANCE ADVANTAGES OVER A NON MU TASK
                         ;AND IS FULLY SUPPORTED ON P/OS)
         T4.LDD=20       ;TASK'S LOAD DEVICE HAS BEEN DISMOUNTED
         T4.PRO=10       ;TCB IS (OR SHOULD BE) A PROTOTYPE
         T4.PRV=4        ;TASK WAS PRIV, BUT HAS CLEARED T3.PRV
                         ;WITH GIN (MAY RESET WITH GIN IF T4.PRV SET)
         T4.DSP=2        ;TASK WAS BUILT FOR USER I/D SPACE
         T4.SNC=1        ;TASK USES COMMONS FOR SYNCHRONIZATION

;+
; REQUIRED RUN MASK
;-

         TR.UBT=100000   ;UNIBUS RUN T
         TR.UBS=40000    ;UNIBUS RUN S
         TR.UBR=20000    ;UNIBUS RUN R
         TR.UBP=10000    ;UNIBUS RUN P
         TR.UBN=4000     ;UNIBUS RUN N
         TR.UBM=2000     ;UNIBUS RUN M
         TR.UBL=1000     ;UNIBUS RUN
         TR.UBK=400      ;UNIBUS RUN K
         TR.UBJ=200      ;UNIBUS RUN J
         TR.UBH=100      ;UNIBUS RUN H
         TR.UBF=40       ;UNIBUS RUN F
         TR.UBE=20       ;UNIBUS RUN E
         TR.CPD=10       ;PROCESSOR D
         TR.CPC=4        ;PROCESSOR C
         TR.CPB=2        ;PROCESSOR
         TR.CPA=1        ;PROCESSOR A

                 .ENDC

                 .PSECT
```

A.24 UCBDF$

```
;+
; 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. A UCB EXISTS FOR EVERY LOGICAL
; UNIT ON THE SYSTEM AND DEFINES UNIT CHARACTERISTICS AS WELL AS
; CURRENT UNIT STATUS.
;
;-
         .ASECT

.=177772

         .IF NB  SYSDEF

         .IF DF  A$$CNT

.=177770

U.UAB:   .BLKW   1       ;POINTER TO USER ACCOUNT BLOCK (DV.TTY ONLY)

         .IFF

U.UAB:

         .ENDC

         .ENDC

177772   U.MUP:  .BLKW   1 ; (-6) MULTI-USER PROTECTION WORD (DV.TTY
                           ; ONLY)
177774   U.LUIC: .BLKW   1 ;(-4) LOGIN UIC (DV.TTY ONLY)
177776   U.OWN:  .BLKW   1 ;(-2) OWNING TERMINAL(DEVICE ALLOCATION)
000000   U.DCB:  .BLKW   1 ;BACK POINTER TO DCB
000002   U.RED:  .BLKW   1 ;POINTER TO REDIRECT UNIT UCB
000004   U.CTL:  .BLKB   1 ;CONTROL PROCESSING FLAGS
000005   U.STS   .BLKB   1 ;UNIT STATUS
000006   U.UNIT: .BLKB   1 ;PHYSICAL UNIT NUMBER
000007   U.ST2:  .BLKB   1 ;UNIT STATUS EXTENSION
000010   U.CW1:  .BLKW   1 ;FIRST DEVICE CHARACTERISTICS WORD
000012   U.CW2:  .BLKW   1 ;SECOND DEVICE CHARACTERISTICS WORD
000014   U.CW3:  .BLKW   1 ;THIRD DEVICE CHARACTERISTICS WORD
000016   U.CW4:  .BLKW   1 ;FOURTH DEVICE CHARACTERISTICS WORD
000020   U.SCB:  .BLKW   1 ;POINTER TO SCB
```

UCBDF$

```
000022    U.ATT:  .BLKW   1 ;TCB ADDRESS OF ATTACHED TASK
000024    U.BUF:  .BLKW   1 ;RELOCATION BIAS OF CURRENT I/O REQUEST
000026            .BLKW   1 ;BUFFER ADDRESS OF CURRENT I/O REQUEST
000030    U.CNT:  .BLKW   1 ;BYTE COUNT OF CURRENT I/O REQUEST
                            ;(DV.MSD)
000032    U.UCBX=U.CNT+2    ;BIAS OF UCB EXTENSION IN SEC POL
000034    U.ACP=U.CNT+4     ;ADDRESS OF TCB OF MOUNTED ACP
000036    U.VCB=U.CNT+6     ;ADDRESS OF VOLUME CONTROL BLOCK
000032    U.CBF=U.CNT+2     ;CONTROL BUFFER RELOCATION AND ADDRESS
000040    U.UMB=U.CNT+10    ;ADDRESS OF UMB FOR SHADOW RECORDING
000042    U.PRM=U.CNT+12    ;DISK SIZE PARAMETER WORDS
000046    U.UTMO=U.CNT+16   ;UNIT COMMAND TIME OUT
000050    U.LHD=U.CNT+20    ;UNIT OUTSTANDING I/O PACKET LISTHEAD
000046    U.ICSR=U.CNT+16   ;CSR ADDRESS (P/OS V1 DRIVER ACCESS ONLY)
000050    U.SLT=U.CNT+20    ;SLOT NUMBER (P/OS V1 DRIVER ACCESS ONLY)
000052    U.SPRM=U.CNT+22   ;4 WD SAVED I/O PACKET AREA (DV.MSD AND
                            ;USAGE

                  ;OF $VOLSC) USED AT I/O COMPLETION TO RESTORE
                  ;I/O PACKET IF STALLED I/O CONDITION IN EFFECT)

000054    U.BPKT=U.CNT+24   ;UNIT BAD BLOCK PACKET WAITING LIST
000060    U.UC2X=U.CNT+30   ;POINTER TO SECOND EXTENSION IN
                            ;SECONDARY POOL
000062    U.OTRF=U.CNT+32   ;OUTSTANDING COMMAND STATUS REQUEST
                            ;REGISTER
000064    U.CMST=U.CNT+34   ;COMMAND STATUS PROGRESS REGISTER


                  ;
                  ; MAGTAPE DEVICE DEPENDANT UCB OFFSETS
                  ;

000040    U.SNUM=U.CNT+10   ;SLAVE UNIT NUMBER
000042    U.FCDE=U.CNT+12   ;FUNCTION CODE
000044    U.KRB1=U.CNT+14   ;SUBCONTROLLER KRB1 POINTER
                            ;

          ;
          ; DEFINE SECONDARY POOL UCB EXTENSION OFFSETS
          ; (DV.MSD DEVICES ONLY)
          ;
          .=0
000000            .BLKW   9.;FIXED ACCOUNTING TRANSACTION HEADER
000022    X.NAME: .BLKW   2 ;DRIVE NAME IN RADIX-50
000026    X.IOC:  .BLKW   2 ;I/O COUNT
000032    X.ERHL: .BLKB   1 ;HARD ERROR LIMIT
000033    X.ERSL: .BLKB   1 ;SOFT ERROR LIMIT
000034    X.ERSC: .BLKB   1 ;SOFT ERROR COUNT
000035    X.ERHC: .BLKB   1 ;HARD ERROR COUNT
000036    X.WCNT: .BLKW   2 ;WORDS TRANSFERED COUNT
```

UCBDF$

```
        ;
        ; DEFINE OFFSETS FOR SEEK OPTIMIZATION DEVICES
        ;

000042  X.CYLC: .BLKW   2 ;CYLINDERS CROSSED COUNT
000046  X.CCYL: .BLKW   1 ;CURRENT CYLINDER
000050  X.FCUR: .BLKB   1 ;CURRENT FAIRNESS COUNT
000051  X.FLIM:           ;FAIRNESS COUNT LIMIT
000051  X.DSKD: .BLKB   1 ;DISK DIRECTION (HIGH BIT 1=OUT)

000052  X.DNAM: .BLKW   1 ;DEVICE NAME FOR ACCOUNTING
000054  X.UNIT: .BLKB   1 ;UNIT NUMBER FOR ACCOUNTING
000055          .BLKB   1 ;UNUSED FOR NOW
000056  X.LGTH=.          ;LENGTH OF THE UCB EXTENSION
000012  X.DFFL=10.        ;DEFAULT FAIRNESS COUNT LIMIT
000010  X.DFSL=8.         ;DEFAULT SOFT ERROR LIMIT
000005  X.DFHL=5.         ;DEFAULT HARD ERROR LIMIT

        ;
        ; DEFINE OFFSETS FOR DISK MSCP CONTROLLERS (SECOND UCB
        ; EXTENSION)
        ;

        ;
        ; CHARACTERISTICS OBTAINED FROM "GET UNIT STATUS" END
        ; PACKETS
        ;

        .=0
000000  X.MLUN: .BLKW   1 ;MULTI-UNIT CODE
000002  X.UNFL: .BLKW   1 ;UNIT FLAGS
000004          .BLKW   2 ;RESERVED
000010  X.UNTI: .BLKW   4 ;UNIT IDENTIFIER
000020  X.MEDI: .BLKW   2 ;MEDIA IDENTIFIER
000024  X.SHUN: .BLKW   1 ;SHADOW UNIT
000026  X.SHST: .BLKW   1 ;SHADOW UNIT STATUS
000030  X.TRCK: .BLKW   1 ;UNIT TRACK SIZE
000032  X.GRP:  .BLKW   1 ;UNIT GROUP SIZE
000034  X.CYL:  .BLKW   1 ;UNIT CYLINDER SIZE
000036  X.USVR: .BLKB   1 ;UNIT SOFTWARE VERSION
000037  X.UHVR: .BLKB   1 ;UNIT HARDWARE VERSION
000040  X.RCTS: .BLKW   1 ;UNIT RCT TABLE SIZE
000042  X.RBNS: .BLKB   1 ;UNIT RBN 'S / TRACK
000043  X.RCTC: .BLKB   1 ;UNIT RCT COPIES
        ;
        ; CHARACTERISTICS OBTAINED FROM "ONLINE" OR "SET UNIT
        ; CHARACTERISTICS" END PACKETS
        ;
000044  X.UNSZ: .BLKW   2 ;UNIT SIZE
000050  X.VSER: .BLKW   2 ;VOLUME SERIAL NUMBER
```

```
000054   X.DUSZ=.           ;SIZE OF DISK MSCP CONTROLLER UCB
                            ; EXTENSION


                .IF NB  TTDEF

         ;
         ; TERMINAL DRIVER DEFINITIONS
         ;
         .=U.BUF
000024   U.TUX:  .BLKW   1 ;POINTER TO UCB EXTENSION (UCBX)
000026   U.TSTA: .BLKW   4 ;STATUS QUADRUPLE-WORD
000036   U.UIC:  .BLKW   1 ;DEFAULT UIC <====(ANY DV.TTY DEVICE)
000040   U.TLPP: .BLKB   1 ;LINES PER PAGE
000041   U.TFRQ: .BLKB   1 ;FORK REQUEST BYTE
000042   U.TFLK: .BLKW   1 ;FORK LIST LINK WORD
000044   U.TCHP: .BLKB   1 ;CURRENT HORIZONTAL POSITION
000045   U.TCVP: .BLKB   1 ;CURRENT VERTICAL POSITION
000046   U.TTYP: .BLKB   1 ;TERMINAL TYPE
000047   U.TMTI: .BLKB   1 ;MODEM TIMER
000050   U.TTAB: .BLKW   1 ;IF 0: U.TTAB+1 IS SINGLE-CHARACTER
                           ;    TYPEAHEAD BUFFER, CURRENTLY EMPTY
                           ;IF ODD: U.TTAB+1 IS SINGLE-CHARACTER TYPE-
                           ;    AHEAD BUFFER AND HOLDS A CHARACTER
                           ;IF NON-0 AND EVEN: POINTER TO MULTI-
                           ;    CHARACTER TYPEAHEAD BUFFER
         .=.-2             ;THE NEXT TWO OFFSETS OVERLAP U.TTAB WHEN
                           ;    THE TYPEAHEAD BUFFER IS IN
                           ;    SECONDARY POOL
000050   U.TECO: .BLKB   1 ;ECHO BUFFER FOR DMA OPERATIONS WHEN UCBX
                           ;    IS SECONDARY POOL AND THUS NOT
                           ;    MAPPED BY A UMR
000051   U.TBSZ: .BLKB   1 ;TYPEAHEAD BUFFER SIZE
000052   U.ACB:  .BLKW   1 ;ANCILLARY CONTROL DRIVER BLOCK ADDR
000054   U.AFLG: .BLKW   1 ;ANCILLARY CONTROL DRIVER FLAGS WORD
000056   U.ADMA: .BLKW   1 ;ANCILLARY CONTROL DRIVER DMA BUFFER

         ;
         ; DEFINE BITS IN STATUS WORD 1 (U.TSTA)
         ;
         S1.RST=1          ;READ WITH SPECIAL TERMINATORS IN PROGRESS
         S1.RUB=2          ;RUBOUT SEQUENCE IN PROGRESS (NON-SCOPE)
         S1.ESC=4          ;ESCAPE SEQUENCE IN PROGRESS
         S1.RAL=10         ;READ ALL IN PROGRESS
         S1.RNE=20         ;ECHO SUPPRESSED
         S1.CTO=40         ;OUTPUT STOPPED BY CTRL-O
         S1.OBY=100        ;OUTPUT BUSY
         S1.IBY=200        ;INPUT BUSY
         S1.BEL=400        ;BELL PENDING
         S1.DPR=1000       ;DEFER PROCESSING OF CHAR. IN U.TECB
         S1.DEC=2000       ;DEFER ECHO OF CHAR. IN U.TECB
```

```
S1.DSI=4000         ;INPUT PROCESSING DISABLED
S1.CTS=10000        ;OUTPUT STOPPED BY CTRL-S
S1.USI=20000        ;UNSOLICITED INPUT IN PROGRESS
S1.OBF=40000        ;BUFFERED OUTPUT IN PROGRESS
S1.IBF=100000       ;BUFFERED INPUT IN PROGRESS

;
; DEFINE BITS IN STATUS WORD 2 (U.TSTA+2)
;
S2.ACR=1            ;WRAP-AROUND (AUTOMATIC CR-LF) REQUIRED
S2.WRA=6            ;CONTEXT FOR WRAP-AROUND
S2.WRB=2            ;LOW BIT IN S2.WRA BIT PATTERN
S2.CR=10            ;TRAILING CR REQUIRED ON OUTPUT
S2.BRQ=20           ;BREAK-THROUGH-WRITE REQUEST IN QUEUE
S2.SRQ=40           ;SPECIAL REQUEST IN QUEUE
                    ;    (IO.ATT, IO.DET, SF.SMC)
S2.ORQ=100          ;OUTPUT REQUEST IN QUEUE (MUST = S1.OBY)
S2.IRQ=200          ;INPUT REQUEST IN QUEUE (MUST = S1.IBY)
S2.HFL=3400         ;HORIZONTAL FILL REQUIREMENT
S2.VFL=4000         ;VERTICAL FILL REQUIREMENT
S2.HHT=10000        ;HARDWARE HORIZONTAL TAB PRESENT
S2.HFF=20000        ;HARDWARE FORM-FEED PRESENT
S2.FLF=40000        ;FORCE LINE FEED BEFORE NEXT ECHO
S2.FDX=100000       ;LINE IS IN FULL DUPLEX MODE

;
; DEFINE BITS IN STATUS WORD 3 (U.TSTA+4)
;
S3.RAL=10           ;TERMINAL IS IN READ-PASS-ALL MODE
                    ;(S3.RAL MUST = S1.RAL)
S3.RPO=20           ;READ W/PROMPT OUTPUT IN PROGRESS
S3.WES=40           ;TASK WANTS ESCAPE SEQUENCES
S3.TAB=100          ;TYPEAHEAD BUFFER ALLOCATION REQUESTED
S3.8BC=200          ;PASS 8 BITS ON INPUT
S3.RCU=400          ;RESTORE CURSOR (MUST = TF.RCU*400)
S3.ABD=1000         ;AUTO-BAUD SPEED DETECTION ENABLED
S3.ABP=2000         ;AUTO-BAUD SPEED DETECTION IN PROGRESS
S3.WAL=4000         ;WRITE-PASS-ALL (MUST = TF.WAL*400)
S3.VER=10000        ;LAST CHAR. IN TYPEAHEAD BUFFER
                    ;HAS PARITY ERROR
S3.BCC=20000        ;LAST CHAR. IN TYPEAHEAD BUFFER
                    ;HAS FRAMING ERROR
S3.DAO=40000        ;LAST CHAR. IN TYPEAHEAD BUFFER
                    ;HAS DATA OVERRUN ERROR
                    ;NOTE - THE 3 BITS ABOVE MUST CORRESPOND
                    ;TO THE RESPECTIVE ERROR FLAGS IN THE
                    ;HARDWARE RECEIVE BUFFER
S3.PCU=100000       ;POSITION CURSOR BEFORE WRITE

;
; DEFINE BITS IN STATUS WORD 4 (U.TSTA+6)
```

```
        ;
        S4.INT=40            ;^C INT-DO HANDLED DIFFERENTLY FOR
                             ;IO.RTT ON P/OS
        S4.DLO=100           ;DIAL-OUT LINE (IMPLIES U2.RMT)
        S4.ANI=400           ;ANSI CRT TERMINAL
        S4.AVO=1000          ;VT100-FAMILY TERMINAL DISPLAY
        S4.BLK=2000          ;BLOCK MODE TERMINAL
        S4.DEC=4000          ;DIGITAL CRT TERMINAL
        S4.EDT=10000         ;TERMINAL HAS LOCAL EDITING FUNCTIONS
        S4.RGS=20000         ;TERMINAL SUPPORTS REGIS GRAPHICS
        S4.CTC=40000         ;TERMINAL WANTS CLI TO HAVE ^C
                             ;NOTIFICATION
                             ;

                .ENDC

                ;
                ; VIRTUAL TERMINAL UCB DEFINITIONS
                ;

                .=U.UNIT
000006  U.OCNT: .BLKB   1 ;OFFSPRING WITH THIS AS TI:
        .=U.BUF
000024  U.RPKT: .BLKW   1 ;CURRENT OFFSPRING READ I/O PACKET
000026  U.WPKT: .BLKW   1 ;CURRENT OFFSPRING WRITE I/O PACKET
000030  U.IAST: .BLKW   1 ;INPUT AST ROUTINE ADDRESS
000032  U.OAST: .BLKW   1 ;OUTPUT AST ROUTINE ADDRESS
000034  U.AAST: .BLKW   1 ;ATTACH AST ROUTINE ADDRESS


                .IF NB  TTDEF

        .IIF NE U.AAST+2-U.UIC .ERROR  ;ADJACENCY ASSUMED

                .ENDC


        .=U.AAST+4
000040  U.PTCB:.BLKW    1 ;PARENT TCB ADDRESS

        ;
        ; CONSOLE DRIVER DEFINITIONS
        ;
        .=U.BUF+2
000026  U.CTCB: .BLKW   1 ;ADDRESS OF CONSOLE LOGGER TCB
000030  U.COTQ: .BLKW   2 ;I/O PACKET LIST QUEUE
000034  U.RED2: .BLKW   1 ;REDIRECT UCB ADDRESS


                .PSECT
```

UCBDF$

```
;+
; DEVICE TABLE STATUS DEFINITIONS
;
; DEVICE CHARACTERISTICS WORD 1 (U.CW1) DEVICE TYPE
; DEFINITION
; BITS.
;-
        DV.REC=1          ;RECORD ORIENTED DEVICE (1=YES)
        DV.CCL=2          ;CARRIAGE CONTROL DEVICE (1=YES)
        DV.TTY=4          ;TERMINAL DEVICE (1=YES)
        DV.DIR=10         ;FILE STRUCTURED DEVICE (1=YES)
        DV.SDI=20         ;SINGLE DIRECTORY DEVICE (1=YES)
        DV.SQD=40         ;SEQUENTIAL DEVICE (1=YES)
        DV.MSD=100        ;MASS STORAGE DEVICE (1=YES)
        DV.UMD=200        ;USER MODE DIAGNOSTICS SUPPORTED (1=YES)
        DV.MBC=400        ;RESERVED
        DV.EXT=400        ;RESERVED
        DV.SWL=1000       ;UNIT SOFTWARE WRITE LOCKED (1=YES)
        DV.ISP=2000       ;INPUT SPOOLED DEVICE (1=YES)
        DV.OSP=4000       ;OUTPUT SPOOLED DEVICE (1=YES)
        DV.PSE=10000      ;PSEUDO DEVICE (1=YES)
        DV.COM=20000      ;RESERVED
        DV.F11=40000      ;DEVICE IS MOUNTABLE AS F11 DEVICE (1=YES)
        DV.MNT=100000     ;DEVICE IS MOUNTABLE (1=YES)

;+
; TERMINAL (DV.TTY)DEPENDENT CHARACTERISTICS WORD 2
; (U.CW2) BIT DEFINITIONS
;-
        U2.DH1=100000     ;UNIT IS A MULTIPLEXER (1=YES)
        U2.DJ1=40000      ;UNIT IS A DJ11 (1=YES)
        U2.RMT=20000      ;UNIT IS REMOTE (1=YES)
        U2.HFF=10000      ;UNIT DOES HRDWRE FORM FEEDS(1=YES)
        U2.L8S=10000      ;OLD NAME FOR U2.HFF
        U2.NEC=4000       ;DON'T ECHO SOLICITED INPUT (1=YES)
        U2.CRT=2000       ;UNIT IS A CRT (1=YES, ANY DV.TTY IMPLIES
                          ;(DEL=BSP,SPC,BSP AND TERM TYP INDEPENDENT
                          ;    CURS POSITIONING))
        U2.ESC=1000       ;UNIT GENERATES ESCAPE SEQ.(1=YES,ANY DV.TTY)
        U2.LOG=400        ;USER LOGGED ON TERMINAL (0=YES) (ANY DV.TTY)
        U2.SLV=200        ;UNIT IS SLAVE TERM(1=YES) (RESRVD ANY
                         ;DV.TTY)
        U2.DZ1=100        ;UNIT IS A DZ11 (1=YES)
        U2.HLD=40         ;TERMINAL IS IN HOLD SCREEN MODE (1=YES)
                          ;    (VT52)
        U2.AT.=20         ;(RESERVED)
        U2.PRV=10         ;UNIT IS A PRIVILEGED TERM.(1=YES, ANY
                         ;DV.TTY)
        U2.L3S=4          ;UNIT IS A LA30S TERMINAL (1=YES)
        U2.VT5=2          ;UNIT IS A VT05B TERMINAL (1=YES)
```

```
        U2.LWC=1            ;LOWER CASE TO UPPER CASE CONVERSION ON INPUT
                            ;(0=YES)

;+
; BIT DEFINITIONS FOR U.MUP (NOT USED CURRENTLY, DV.TTY ONLY)
;-
        UM.OVR=1            ;OVERRIDE CLI INDICATOR
        UM.CLI=36           ;CLI INDICATOR BITS
        UM.DSB=200          ;TERMINAL DISABLED SINCE CLI ELIMINATED
        UM.NBR=400          ;NO BROADCAST
        UM.CNT=1000         ;CONTINUATION LINE IN PROGRESS
        UM.CMD=2000         ;COMMAND IN PROGRESS
        UM.SER=4000         ;SERIAL COMMAND RECOGNITION ENABLED
        UM.KIL=10000        ;TTDRV SHOULD SEND KILL PKT ON CNTRL/C


;+
; TERMINAL SECONDARY POOL OFFSETS FOR THE UCB EXTENSION AND
; TYPEAHEAD BUFFER
;-
        U.TAPR=24           ;OFFSET WITHIN UCB WHICH POINTS TO UCB
                            ;    EXTENSION
        U.TTBF=46           ;OFFSET WITHIN UCB EXTENSION WHICH POINTS
                            ;    TO TYPEAHEAD BUFFER

;+
; TERMINAL DEPENDENT CHARACTERISTICS WORD 3 (U.CW3) BIT
; DEFINITIONS
;-
        U3.UPC=20000        ;UPCASE OUTPUT FLAG
        U3.PAR=40000        ;PARITY GENERATION AND CHECKING
        U3.OPA=100000       ;PARITY SENSE (1=ODD PARITY)

;+
; VIRTUAL TERMINAL 3RD CHARACTERISTICS WORD DEFINITIONS
; (DRIVER SPECIFIC)
;-
        U3.FDX=1            ;FULL DUPLEX MODE (1=YES)
        U3.DBF=2            ;INTERMEDIATE BUFFERING DISABLED (1=YES)
        U3.RPR=4            ;READ W/PROMPT IN PROGRESS (1=YES)

;+
; TERMINAL DEPENDENT CHARACTERISTICS WORD 4 (U.CW4) BIT DEF.
; (DRIVER SPECIFIC)
;-
        U4.CR=100           ;LOOK FOR CARRIAGE RETURN

;+
; UNIT CONTROL PROCESSING FLAG DEFINITIONS
;-
        UC.ALG=200          ;BYTE ALIGNMENT ALLOWED (1=NO)
```

```
        UC.NPR=100          ;DEVICE IS AN NPR DEVICE (1=YES)
        UC.QUE=40           ;CALL DRIVER BEFORE QUEUING (1=YES)
        UC.PWF=20           ;CALL DRIVER AT POWERFAIL ALWAYS (1=YES)
        UC.ATT=10           ;CALL DRIVER ON ATTACH/DETACH (1=YES)
        UC.KIL=4            ;CALL DRIVER AT I/O KILL ALWAYS (1=YES)
        UC.LGH=3            ;TRANSFER LENGTH MASK BITS

;+
; UNIT STATUS BIT DEFINTIONS
;-
        US.BSY=200          ;UNIT IS BUSY (1=YES)
        US.MNT=100          ;UNIT IS MOUNTED (0= YES)<=CAREFUL OF
                            ;    POLARITY!
        US.FOR=40           ;UNIT IS MOUNTED AS FOREIGN VOLUME (1=YES)
        US.LAB=4            ;UNIT HAS LABELED TAPE ON IT (1=YES)
                            ;(HAS MEANING FOR DV.MNT AND
                            ;    (US.MNT!US.FOR=0))
        US.MDM=20           ;UNIT IS MARKED FOR DISMOUNT (1=YES)
        US.PWF=10           ;POWERFAIL OCCURED (1=YES).

;+
; CARD READER DEPENDENT UNIT STATUS BIT DEFINITIONS
;-
        US.ABO=1            ;UNIT IS MARKED FOR ABORT IF NOT READY
                            ;    (1=YES)
        US.MDE=2            ;UNIT IS IN 029 TRANSLATION NODE (1=YES)

;+
; DV.MSD DEPENDENT UNIT STATUS BITS
;-
        US.WCK=10           ;WRITE CHECK ENABLED (1=YES)
        US.SPU=2            ;UNIT IS SPINNING UP (1=YES)
        US.VV=1             ;VOLUME VALID IS SET (1=YES)

;+
; TERMINAL DEPENDENT UNIT STATUS BIT DEFINITIONS
;-
        US.CRW=4            ;UNIT IS WAITING FOR CARRIER (1=YES)
        US.DSB=2            ;UNIT IS DISABLED (1=YES)
        US.OIU=1            ;OUTPUT INTERRUPT IS UNEXPECTED ON UNIT
                            ;    (1=YES)

;+
; UNIT STATUS EXTENSION (U.ST2) BIT DEFINITIONS
;-
        US.OFL=1            ;UNIT OFFLINE (1=YES)
        US.RED=2            ;UNIT REDIRECTABLE (0=YES)
        US.PUB=4            ;UNIT IS PUBLIC DEVICE (1=YES)
        US.UMD=10           ;UNIT ATTACHED FOR DIAGNOSTICS (1=YES)
        US.PDF=20           ;PRIVILEGED DIAGNOSTIC FUNCTIONS ONLY
                            ;    (1=YES)
```

```
        US.MUN=40           ;MULTI-UNIT FLAG
        US.TRN=100          ;UNIT TRANSITION HAS OCCURRED (1=YES)
        US.SIO=200          ;STALL I/O TO UNIT (1=YES)  (ANY DEVICE)

;+
; MAGTAPE DENSITY SUPPORT DEFINITION IN U.CW3
;-
        UD.UNS=0            ; UNSUPPORTED
        UD.200=1            ; 200BPI, 7 TRACK
        UD.556=2            ; 556BPI, 7 TRACK
        UD.800=3            ; 800BPI, 7 OR 9 TRACK
        UD.160=4            ; 1600BPI, 9 TRACK
        UD.625=5            ; 6250BPI, 9 TRACK

        .ENDM
```

APPENDIX B

TASK BUILDING AND CLUSTER LIBRARIES

This appendix provides an overview and examples of conceptual and detailed information on overlaying task structures and cluster libraries for PDP-11 and RSX systems. Note that the information is generic. For more detail, also see the Task Builder Manual.

B.1 AN OVERVIEW OF OVERLAYING

This discussion of overlaying covers a complex subject. The *RSX-11M/M-PLUS and Micro/RSX Task Builder Manual* provides more detail on the material covered here, plus information on the extended overlaying facilities that involve mapping via the PLAS directives and memory-resident libraries.

The complexity of task structure on RSX and related PDP-11 systems reflects both the utility and the limitations of the system hardware and software. Over the years, users have found the PDP-11 a suitable base for increasingly ambitious applications. The sizes of these applications have grown far beyond the hardware's 16-bit (64 Kbyte) direct addressing capabilities. Therefore, task structure extensions have evolved to meet the increased addressing needs of such applications in as transparent a manner as possible.

The focus for these extensions has been the PDP-11 Task Builder (TKB), which is a powerful, complex utility. The TKB utility is sometimes criticized for its complexity. Note, however, that it must support run-time transparency (by and large its mechanisms must be transparent to code execution), structural transparency (by and large its structures must continue to support flexible handling of program sections), and performance requirements (whatever it does should not bring the task to its knees). Obviously, providing these level of support will require some assistance from the program's designer. The various DIGITAL products (e.g., language object-time systems, data management services like RMS and FCS, and the Forms Management System) each provide pre-packaged task configuration aids designed to make task-building as automated as possible. If application size requires further packaging efforts, however, the user must become more actively involved in this process. The following overview supplements the detailed descriptions of overlaying in the *RSX-11M/M-PLUS and Micro/RSX Task Builder Manual.*

B.1.1 Basic Overlay Concepts and Constraints

When total program size grows beyond the virtual addressing limits of the task "envelope," some tradeoffs must be made. Arrangements must be made to allow currently-executing code and currently-accessed data to reside in virtual memory while currently-unneeded code and data reside elsewhere (typically on disk), and to allow the configuration of resident segments to change dynamically with changing demands.

One way to accomplish the required tradeoff is with a "code cache" (in this discussion, data segments are included with program code, since even if are separated, they are treated analagously). Also, presume that developers generate code/data segments that are sufficiently small and modular to be easily moved, vector any external CALLs or data references into control structures that specify the module required, and finally, create a wholly-transparent software "paging" scheme along the lines of VAX hardware paging.

Examining some of the difficulties with this approach will help explain the mechanism we actually use:

1. The major problem is the data references. They cannot be "caught" without interpreting the entire instruction stream in software. This tends to defeat the purpose of the PDP-11 instruction set. Also, despite the advantages of "clean" inter-module data linkages, they shouldn't be absolutely mandatory. If they were, it would be necesary for us to detect cases in which pointers to external data (in data structures, the general registers, etc.) are passed as input arguments, and vector these references as well (possibly through multiple levels of indirection).

   The performance overhead (perhaps 100-to-1) of software instruction interpretation rules out mandatory "clean" inter-module data linkages. Also, the goal of reasonable coding transparency rules out the extremely strict rules on inter-module data references that we would need without such interpretation. The conclusion is that data, when present, must always appear at the same virtual address.

   If the software supports the "PC-relative" hardware addressing modes for external data, then code, when present, also must appear at the same virtual address. In fact, even without external data references it could be awkward for code to move around - consider the case of nested subroutines with RETURN addresses on the stack.

2. Having constant data addresses solves only half the interpretation problem. Structure conventions are needed, perhaps supported by semi-automated mechanisms, to ensure that the desired data is actually resident in virtual memory when it is referred to by an instruction. The instruction itself won't be able to verify this.

   Any conventions, of course, limit our flexibility for configuring overlaid data. To allow for special cases, we should consider providing an escape mechanism so that the user code can explicitly request a data segment be loaded prior to referring to it.

3. Normal external CALL and JMP transfers of control can be intercepted fairly easily. TKB must resolve the destinations, and can route them through a loading routine that preserves the general registers and stack depth while it demand-loads the target code segment if it is not already resident.

   Given the lack of "save/restore condition codes" instructions in the hardware, trying to preserve them across CALL/JMP transfers of control would be difficult, and would increase instruction overhead during the transfer. Transfers should be as efficient as possible when the code segment is already resident. Since passing condition code information as an input argument to an external routine is fairly unusual, it should be viewed as a coding restriction.

   Returning condition codes as output from a CALLed routine is not unusual, however, and they should be preserved. The easiest way to ensure their preservation is to require that the CALLing routine remain resident while the target routine executes. In that way, the RETURN requires no special handling.

   Using this approach as described, not only optimizes CALL/RETURN linkage performance, but also solves the problem of how to re-load the CALLer transparently should it be displaced by the target routine. It isn't feasible to keep loading information describing the CALLer on the stack (since stack depth is frequently relevant to the target routine), and would have to maintain a special overlay run-time system stack for this purpose. Also, some routine linkages use the RETURN address as a pointer to in-line input arguments. It would be best not to tamper with it.

4. Other less common transfer-of-control mechanisms include coroutines and tables containing external dispatch addresses. Some of the same issues mentioned above apply to them. It may be reasonable to restrict generality of support for such linkages, but be certain you understand what is entailed.

5. Even creating vectoring information for just the external CALL/JMP references could result in very large vector tables, which in turn would increase pressure on the very virtual memory resources we are trying to conserve. Instead of "paging" on a module-by-module basis, it is better to provide a way of grouping related modules together so that (a) references that occur within the group need not be vectored at all (the group is "paged" as a unit), and (b) a single structure descriptor can serve the entire group (though individual vectors will still be needed for each entry point referred to externally).

6. Another good reason to page related modules as a group is that performance will degrade if many small modules are "paged" individually, as several short disk "read" operations typically take much longer than a single long "read" of equivalent content.

B.1.2 The Overlay Structure

Our main implementation constraints so far are thus:

- To support the way the hardware performs data references in executing instructions. When a given segment of code or data appears in the overlay cache area, it should always appear at the same virtual address.

- To support frequently-used subroutine linkage mechanisms. The CALLing routine should remain resident while the CALLed routine executes.

- To achieve reasonable performance. Groups of related modules should be pageable as single units ("overlay segments"). The vector table overhead is minimized, since purely intra-group references need not be vectored.

B.1.2.1 Overlaying Code Segments - The requirement stated in Section B.1 above that CALLers remain resident in the "code cache" while the target routine executes, plus the desirability of allowing CALLed routines to be nested, means that the logical structure for organizing code segment overlays in virtual memory is a "tree".

With such a tree structure, the "root" of the tree is not overlaid. It permanently resides at a fixed location in the task's virtual address space. Routines in the root may CALL routines in one of the multiple overlay segments in the first level above the root level. (They may of course CALL other routines in the root level as well.) When a Level 1 routine is CALLed in this manner, TKB has actually caused the CALL to be redirected to an "autoload vector" in the root segment. This vector passes a "segment descriptor" address (in the root segment) plus the virtual address of the target routine within that segment to a common "autoload" routine in the root segment's overlay run-time system. This system uses the segment descriptor information to load the segment into virtual memory, if it is not already resident, and then passes control to the real target routine as if it had been CALLed directly.

One autoload vector can service all references from the root to a single Level 1 routine. A single such segment descriptor can service all references to all routines in a single Level 1 overlay segment.

Each routine in Level 1 may in turn CALL routines in the overlay segments directly above its segment in the second level of the structure, as well as routines in its own overlay segment. Each Level 1 to Level 2 CALL is similarly redirected through an appropriate autoload vector, which resides in the Level 1 segment rather than in the root. Therefore, use of valuable non-overlaid virtual memory space in the root will not be affected.

Note that while a routine in the root may CALL routines in any Level 1 segment, a Level 1 routine may CALL only the sub-set of Level 2 routines directly above its segment. Other Level 2 routines (and routines in other Level 1 segments) are inaccessible and invisible to it, because of the "CALLer must remain resident" requirement that led to the tree structure and the rules for its use.

A sample 4-level overlay tree is diagrammed below. Each entity marked with brackets [] (except the root) represents an overlay segment containing potentially multiple routines in multiple modules. The segment names are arbitrary, but useful, for the discussion.

```
Level 3                        [BBA][BBB][BBC]
                                   \  |  /
Level 2      [AA] [AB]         [BA] [BB] [BC] [BD]      [CA] [CB] [CC]
               \  /               \  | |   /              \  |  /
Level 1         [A]                  [B]                    [C]
                   \                  |                    /
Root Level          [non-overlaid portion of the task]
```

Before analyzing transfers of control in this structure, however, it is advisable to look at two possible extensions:

- o The "CALLer must remain resident" rule, when applied to subroutine nesting, implies that whenever we are executing within an overlay segment, all lower segments along the "path" from that segment to the root are also resident. For example, if executing in segment BBC, then BB, B, and of course the root are resident. There is no reason not to allow a routine in segment BBC to CALL routines in BB, B, and the root - and CALL them directly, without vectoring. This is a "down-tree" CALL.

  Such "down-tree" CALLs are allowed, but are conditional. For instance, assume a BBC routine CALLs a BB routine. Before RETURNing to the BBC routine, the BB routine MUST NOT itself CALL "up-tree" (as it would normally be able to do had it not been CALLed from above). If the BB routine CALLed a BBA routine, for example, the BBA routine could successfully RETURN to the BB routine, but when the BB routine then RETURNed to the BBC routine it would find the BBA segment resident instead of the BBC segment. Since the program could not detect this, the program would quickly run into trouble.

  In other words, CALLing up-tree presents no problems, but when CALLing down-tree, watch out for nested up-tree excursions that would violate the "CALLer must remain resident" rule.

- o While it is reasonable to distribute autoload vectors into the overlay segments that actually use them, it is awkward to distribute segment descriptors in the same manner (e.g., with sharable and/or read-only PLAS-mapped memory-resident libraries). Therefore, you should place all segment descriptors in the task root segment. Given their typical size and numbers, they should present no problems.

  Placing task root segments in the segment descriptor allows you to relax the requirement that up-tree CALLs go up just one tree level. You link the segment descriptors into an actual tree structure that mirrors the conceptual overlay tree. Then you modify the autoload routine to load ALL segments on the path between the CALLer and the target. Thus, for example, when a "B" segment routine CALLs a "BBB" segment routine, the CALL is redirected to an autoload vector in the B segment that specifies the actual target virtual address plus the segment descriptor address of the BBB segment. The autoload routine then checks to see if BBB is resident and, if not, loads the BBB segment and works its way down the segment descriptor tree toward the root, loading additional segments as necessary.

Under the assumption that a segment is resident only if all lower segments on the path between it and the root are also resident, this approach should achieve the desired result. However, to guarantee that assumption, the autoload routine must mark all segments that do not share the new path non-resident, even if some of them do not conflict with the virtual memory used by the new path. This ususally has little effect upon the overlay cache, since overlay configurations frequently conflict. Cases to the contrary can often be optimized through use of the co-tree structures discussed below.

The previous description of the basic code-overlaying mechanisms translate into the following:

1. Any "node" (overlay segment) in the tree can "see" and CALL directly all routines in the same node and in any lower nodes that lie on the path between it and the root.

2. Any node in the tree can "see" and CALL indirectly (through an autoload vector) all routines in nodes directly above it in the overlay tree. In our sample tree, a routine in node B could CALL routines in any node whose name begins with B - but no node whose name begins with A or C. A routine in the task root can clearly CALL routines in any node in the structure.

3. Except as discussed in the first two items above, nodes cannot "see" (or CALL) each other. You may normally place routines with the same globally-defined entry point names in nodes that cannot see each other without conflict. However, if a reference to such a duplicated entry point is made by a lower node which can "see" both instances of it, TKB cannot determine which node should be used to satisfy the up-tree CALL and will generate a diagnostic message indicating the ambiguity.

4. Down-tree CALLs should always be carefully examined for nested up-tree CALLs that would displace the CALLer.

5. Any use of overlays at task AST (asynchronous) level also runs a risk. The task-level overlay context could easily be changed "underneath" the running code. Isolating segments used at AST level from those used at task level in separate co-trees is one possible way to avoid this. A few products such as RMS-11 have special asynchronous versions with internal controls that allow mixed task/AST level operation, but most do not.

Wherever a CALL is specified in the above list, a JMP would also be usable.

B.1.2.2 Making the Tree More Flexible

The structure described above works well for applications that fall naturally into tree-structured organization of control flow. Though applications in many cases do lend themselves to such organization, and in many others can be made to comply with minimal changes, there are times when strict tree-structuring imposes significant penalties in use of task virtual address space and/or task image size on disk due to excessive module duplication throughout the structure.

One way to add flexibility to the available control flow mechanisms is to build some explicit extensions into selected portions of the code. For example, suppose that the FOO routine in node BBC of our sample tree would benefit from access to the copy of the BAR routine in node AB of the tree. We could legally just duplicate the BAR routine in node BBC (as long as the root does not CALL BAR, since duplicating BAR would then result in an ambiguous up-tree reference), but if BAR is very large this could be awkward. Instead, however, we could create the following "cross-tree" transfer mechanism:

```
[FOO routine in node BBC]

    FOO::   .               ; (Normal initial FOO code)
            .
            JMP     BAR1    ; (We would normally CALL BAR here)
    FOO2::  .               ; (Remainder of the FOO code)

[Special transfer routine in the root]

    BAR1::  CALL    BAR
            JMP     FOO2
```

The result of this is that FOO legally JMPs to the BAR1 transfer routine (which must be low enough in the tree so that both FOO can see it and it can see BAR), BAR1 legally CALLs BAR, BAR legally RETURNs to BAR1, and BAR1 legally JMPs back to the proper point in FOO. The stack depth has not been affected by this indirection.

For high-level languages that do not permit coding such a sequence directly, one could add a second special MACRO-11 transfer module in the CALLing overlay segment that JMPs to BAR1 and then RETURNs after the root module JMPs back to it, and CALL this second module from the high-level language code. Though such a linkage does not preserve stack depth, due to the added CALL, it will be suitable in many cases (e.g., routines that communicate using the "standard" PDP-11 R5 CALLing sequence).

While inelegant in some ways, the approach as described is a reasonable way to avoid excessive module duplication and reduce virtual address space use by effectively allowing a CALLed routine to displace its CALLer. One must of course be willing to accept the performance overhead of loading the target (plus any non-resident lower nodes) on every CALL and reloading the CALLer (plus any displaced lower nodes) on every RETURN, but for infrequent operations this may well be tolerable. One must also ensure that the target routine does not require access to any data (or code!) resident in the portion of the tree that is displaced when the target is loaded, but again for restricted use this may not be a difficult limitation. Finally, remember that the condition codes are not preserved through the RETURN to FOO in this case.

B.1.2.3 Co-trees - TKB makes another mechanism available for further control over use of the overlay cache area: the co-tree. Conceptually, co-trees allow division of the cache into independent segments that do not affect each other. In effect, they become multiple tree structures, each of which functions independently of the others.

Since overlay activity in one tree does not affect the others, all nodes in each tree can "see" all entry points in all other trees just as if these entry points were in the root segment (though the CALLs must still be vectored through the autoload mechanism). Thus a CALL to a different co-tree is much like a "down-tree" CALL. In particular, you must ensure that any nested CALLs back to the originating tree's code do not change its overlay context such that the CALLer becomes non-resident (the RMS-11 user-provided "get-space" routine feature is an example of such a "CALL-back" situation that must be handled with care).

The Task Builder's normal processing of the default object module library (typically SYSLIB) is designed to avoid occurrences of such inter-co-tree CALL-back situations in cases where their existence would not be obvious due to the implicit nature of the references. The /FULL switch may be used - with caution - to override this behavior when necessary.

Because each co-tree must reserve its full complement of virtual memory all the time, co-tree structures may be somewhat less efficient in use of virtual memory (though more efficient in use of disk space) than equivalent-function single-tree structures with increased module duplication. The Task Builder Manual's descriptions and examples of overlaying provide insights into optimizations and trade-offs among virtual memory, task image size on disk, and performance for single- and multiple-tree structures, as does the RMS-11 "prototype" ODL file RMS11.ODL for specific cases involving the RMS-11 co-tree.

B.1.2.4 Overlaying Data - At the beginning, we decided that intercepting data references would have required on-the-fly interpretation of the instruction stream with prohibitive performance and complexity cost. For this reason, all overlaying of data is left to the user to do correctly.

TKB does, however, provide the tools for overlaying data. Data program sections (PSECTs) can be given the "local" attribute, in which case they will be placed in the overlay segment where the module containing them occurs. As long as such data is referred to only by the code in that segment (or, optionally, by code in segments directly above that segment in the tree structure), all should go well. If referred to by lower segments in the tree, or by segments in other trees, there is no guarantee that the correct data will be resident, and the user is responsible for avoiding such references.

(Data may also occur in-line with code, though this is normally not considered good programming practice. Such data acts much the same as data in a "local" PSECT.)

When a data PSECT is given the "global" attribute, on the other hand, TKB merges all contributions to that PSECT on a given path downward so that they in fact occur in the lowest segment on that path in which any contribution to that PSECT is made. Thus all segments that make contributions to that PSECT may freely refer to it in its entirety without the danger of up-tree data references. Note, however, that there is still no active protection against reference from lower in the tree.

A common programming error involves mixing local and global PSECTs that have the same name. Since the attributes differ, they are treated as different PSECTs, and the local portions are not merged with the global portions.

TKB recognizes use of routine addresses as data just as it recognizes any up-tree CALL-type reference. In a CALL or JMP table, for example, an entry of the form .WORD FOO will be correctly resolved to an autoload vector when the routine is up-tree (or in another tree) from the reference to it. TKB has one unfortunate idiosyncracy in this area: when an up-tree module makes a contribution to a global data PSECT that is forced down-tree to the lowest node in which references to this PSECT occur, any autoload vectors associated with this contribution are placed in the up-tree module's segment rather than in the lower segment in which the reference actually winds up. Thus if a lower node refers to such a contribution, there is no guarantee that the required autoload vector will be resident, and the results can be both surprising and difficult to diagnose. Thankfully, such situations seldom occur in normal use of the overlay facilities.

Finally, TKB allows explicit loading of up-tree data segments via use of the .NAME directive and a "dummy" CALL. This can be useful when large amounts of data must be referred to under program control.

B.2 CLUSTER LIBRARIES AND THEIR USE IN APPLICATIONS

Library clustering retains the performance and ease-of-development advantages associated with use of memory-resident libraries, allows more flexible and efficient use of physical memory by RSX-11M-PLUS and RSTS/E operating systems, and competes favorably with disk overlay structures in use of task virtual memory.

This discussion reviews and supplements the extensive description in the Task Builder Manual, which covers much of this material in slightly different, and sometimes more detailed, form.

B.2.1 Simple Task Structure and Memory Mapping

The 16-bit byte-oriented architecture of the PDP-11 presents the user with 32 KW of addressable memory in the address range 0 to 177777 (octal). In multiprogramming systems, the operating system uses the memory management hardware to map this range of VIRTUAL addresses (as seen by the task) onto one or more ranges of PHYSICAL memory locations, and to ensure that the physical memory associated with each task is suitably protected from unauthorized use by other tasks. This mapping is established through eight hardware Active Page Registers (APRs), each of which controls the mapping and protection of up to 4

KW of task virtual address space, beginning on the appropriate 4 KW virtual address boundary. The APR specifies the start address in physical memory and the size of the task section (both in units of 32-word blocks, the mapping granularity).

The simplest structure is one in which task virtual memory maps onto a continuous series of physical memory locations, as illustrated in Figure B-1. As shown in the figure, the task does not require a full 32 KW of physical memory, and APRs 5, 6, and 7 are set up to reflect the limits of the task memory envelope. Mapping need not be static. If the task is checkpointed to disk to give another task an opportunity to use the physical memory, for example, it may on return occupy a different area in physical memory. If the task requests more space from the operating system, it will acquire access rights to additional physical memory.

The **physical** memory associated with a task may, in fact, exceed 32 KW, under software control of the operating system. The **virtual** memory mapped under hardware control at any instant, however, can never exceed eight "pages," each a maximum of 4 KW in size, and each beginning on a 4 KW virtual address boundary. This ignores systems that support supervisor mode mapping and instruction/data space separation in user tasks - these extend addressable virtual memory through use of the additional APR sets available on PDP-11/70 and PDP-11/44 processors.

For large applications, 32 KW may be inadequate to contain all the necessary code and data for a task. The traditional solution is to **overlay** the task code. This breaks it up into segments, which are loaded on demand from disk and replaced with other segments when no longer needed. Such disk overlaying (which decreases task performance due to the extra disk I/O overhead) is performed under software control of the task's integral overlay run-time system. Overlay segments can be loaded at any word-aligned virtual address boundary. They do not change the virtual-to-physical mapping of the task in any way.

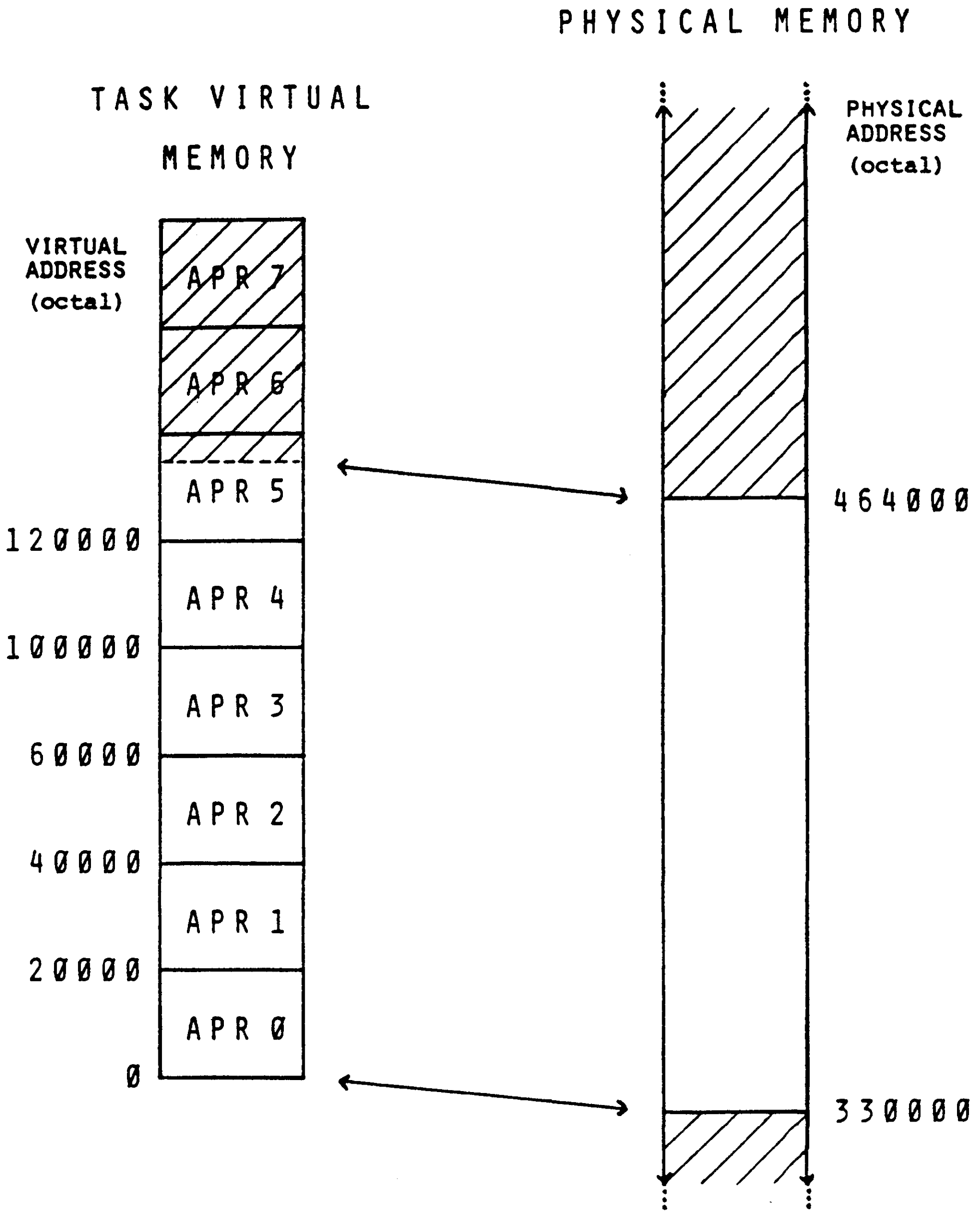


Figure B-1: Simple Task Structure and Memory Mapping

B.2.2 Libraries and Virtual Address Windows

Having eight APRs to work with, a task can map to as many as eight disjoint sections of physical memory (under operating system supervision, of course). Intermediate software structures called a Virtual Address Windows in the system- controlled task header describe each such disjoint section.

The example described in the previous section had the entire task contained in a single continuous section of physical memory, and hence required only a single window (Window 0). As illustrated in Figure B-2, however, Task A has been linked to Library A, a memory-resident library. Since there is no guarantee that the two will be loaded into adjacent sections of physical memory (or that they will have equivalent hardware protection requirements), two virtual address windows are needed. Window 0, as always, maps the task itself, using APRs 0 through 4; the task is permitted both read and write access to this section of memory. Window 1 maps the library using APRs 6 and 7; typical libraries will allow only read access by tasks. Since APR 5 is not currently in use, the task could dynamically extend itself into this virtual address range (libraries are conventionally mapped in the highest available APRs to facilitate this), or could dynamically create an additional virtual address window to map APR 5 to some other portion of physical memory if space for a third window in the task header was specified when the task was built.

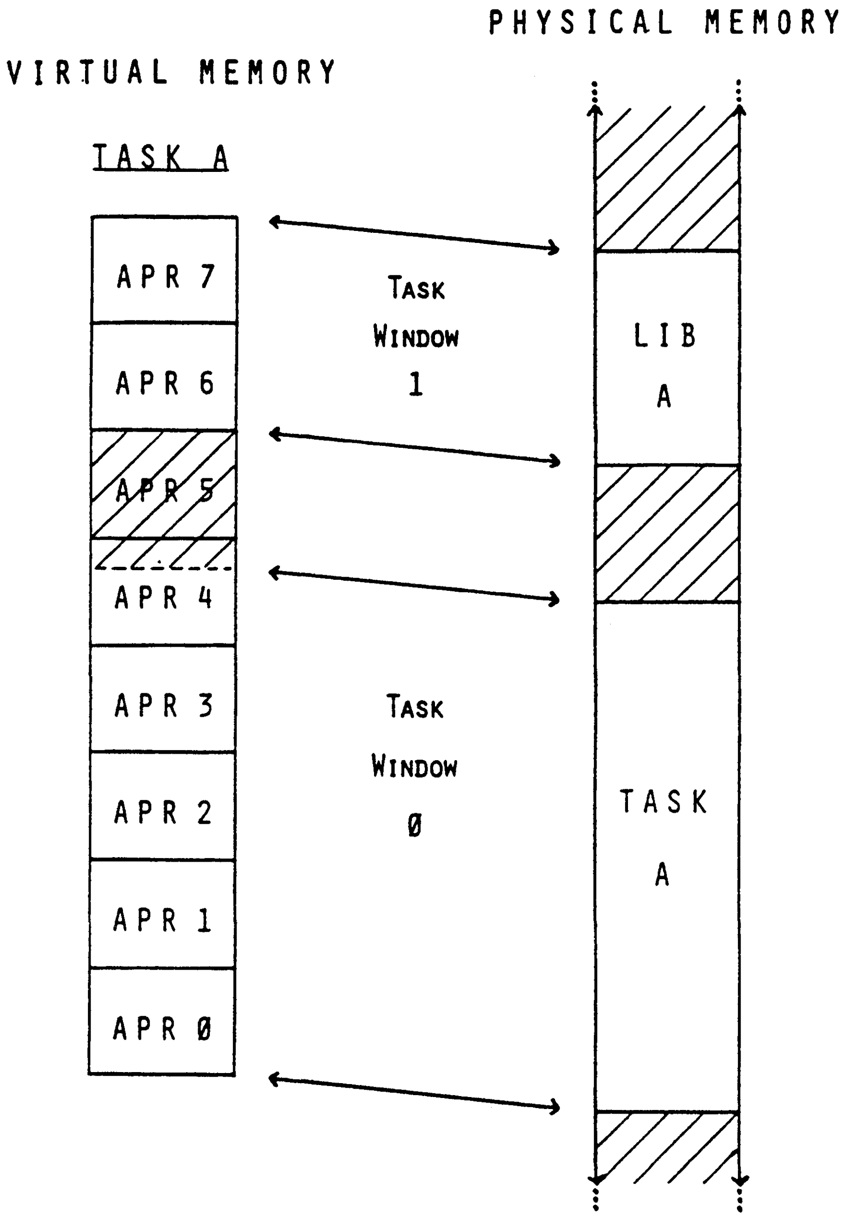


Figure B-2: Libraries and Virtual Address Windows

B.2.3 Library Sharing and Multiple Libraries

Figure B-3 illustrates a more complex (and more typical) case, with multiple tasks and libraries resident in physical memory at the same time.

Task A is still mapped to Library A as before. Task B is also mapped to the same physical copy of Library A. One of the benefits of using sharable re-entrant memory-resident libraries - rather than linking the library code into each task that uses it - is the saving in physical memory use that sharing allows. If the code were resident in the tasks, however, it might be possible to overlay it from disk. This would cut down on per-task physical memory requirements and increase the amount of virtual memory available for the rest of the code in the task (but also decrease performance due to the additional I/O to disk required for task execution).

Whereas Task A maps Library A in APRs 6 and 7 (base virtual address 140000), Task B maps Library A in APRs 4 and 5 (base virtual address 100000). This can be done only if the code in the library is position-independent (PIC - descriptions of position-independent code and re-entrancy can be found in the PDP-11 Processor Handbook). The fact that both tasks use Window 1 to map the library is coincidental.

Task B is also mapped to Library B. Two areas are worthy of note:

1. It is fortunate that Task B is small. The two libraries combined take up fully half the available virtual address space (by using four APRs between them). Task A is too large to be able to map to both libraries simultaneously. If their code were instead disk-overlaid within Task A, there might be room for it, though performance would suffer and the code for Library A would no longer be sharable.

2. Whenever Task B is in memory, both libraries must also be in memory. A region is forced into physical memory whenever an in-memory task maps any portion of it, and Task B cannot run unless 25 KW of physical memory is available - a 9 KW block for the task itself, and an 8 KW block for each library.

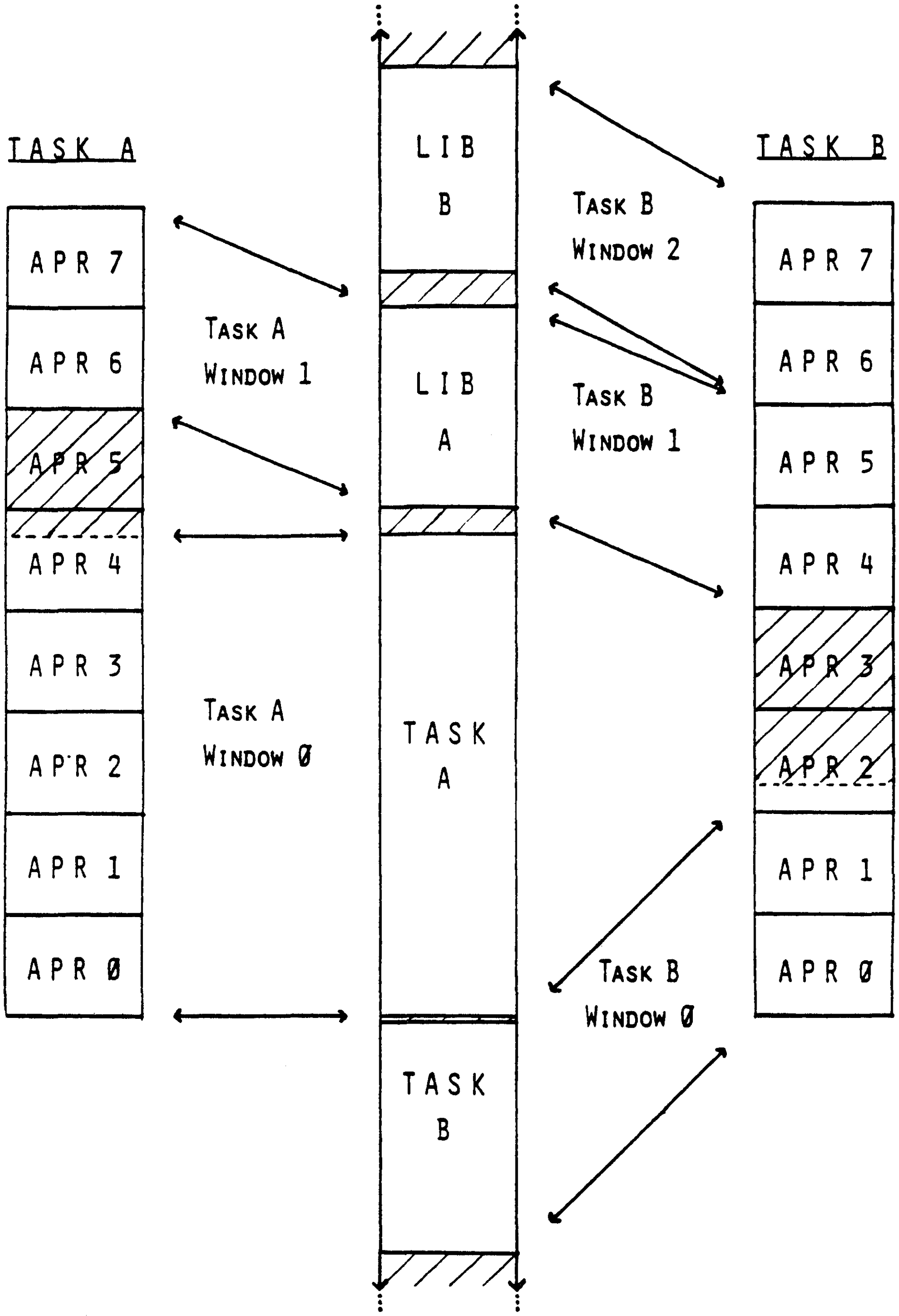


Figure B-3: Library Sharing and Multiple Libraries

B.2.4 Library with Memory-Resident Overlays

The RMS-11 V1.8 full-function memory-resident library RMSRES is an example of code which, for reasons of performance and sharability, is configured as a (re-entrant, position-independent) library but, for reasons of size (about 23 KW), cannot reasonably be mapped "flat-out" by tasks. The Task Builder allows such large libraries to be "resident-overlaid," using the Program Logical Address Space (PLAS) directives to map within the library region. The mapping operation is far less time-consuming than a disk overlay operation. With RMSRES, one APR is continually mapped to common support code in the library "root," and a second APR is used to map whichever library "leaf" segment is currently in use (there are five of these). Even though the library is contained in a single region, the code is mapped in two often discontinuous pieces, therefore an additional window is required in the task.

This approach consumes only 8 KW (two APRs) of the task's virtual address space, while giving the task the benefit of 23 KW of (sharable) support code. This all happens with no disk overlay performance penalty - though disk-overlaid RMS can be contained in about 5 KW per task virtual/physical memory. Despite the fact that no more than 8 KW of the library is mapped by a single task at any one time, the entire library must be resident in physical memory when any portion is mapped by a resident task, since regions cannot be loaded piecemeal.

In Figure B-4, a single task is using both RMSRES (mapping to an arbitrary "leaf" segment is shown) and the 7 KW DIBOL memory-resident library - which is not PLAS-overlaid. Two points are worth noting:

1. The two libraries again consume half the available task virtual address space. Even though the DIBOL library is smaller than 8 KW, the fact that it exceeds 4 KW means that two APRs are needed to map it. While the physical memory required is 7 KW, the virtual memory impact on the task is 8 KW. The "extra" 1 KW at the "top" of APR 5 is not available for use by the task. No other task window can be mapped to it, as it does not begin on an APR (4 KW) address boundary. Similar smaller unusable virtual address ranges can be found at the top of both of the APRs (6 and 7) used to map into RMSRES. The unused 1 KW at the top of APR 3, however, could be used by the task if necessary. It is continuous with the task region, and task Window 0 could be extended to include it.

2. To run at all, the task requires 45 KW of physical memory (15 KW for the task itself, 23 KW for RMSRES, and 7 KW for DIBOL, in continuous blocks).

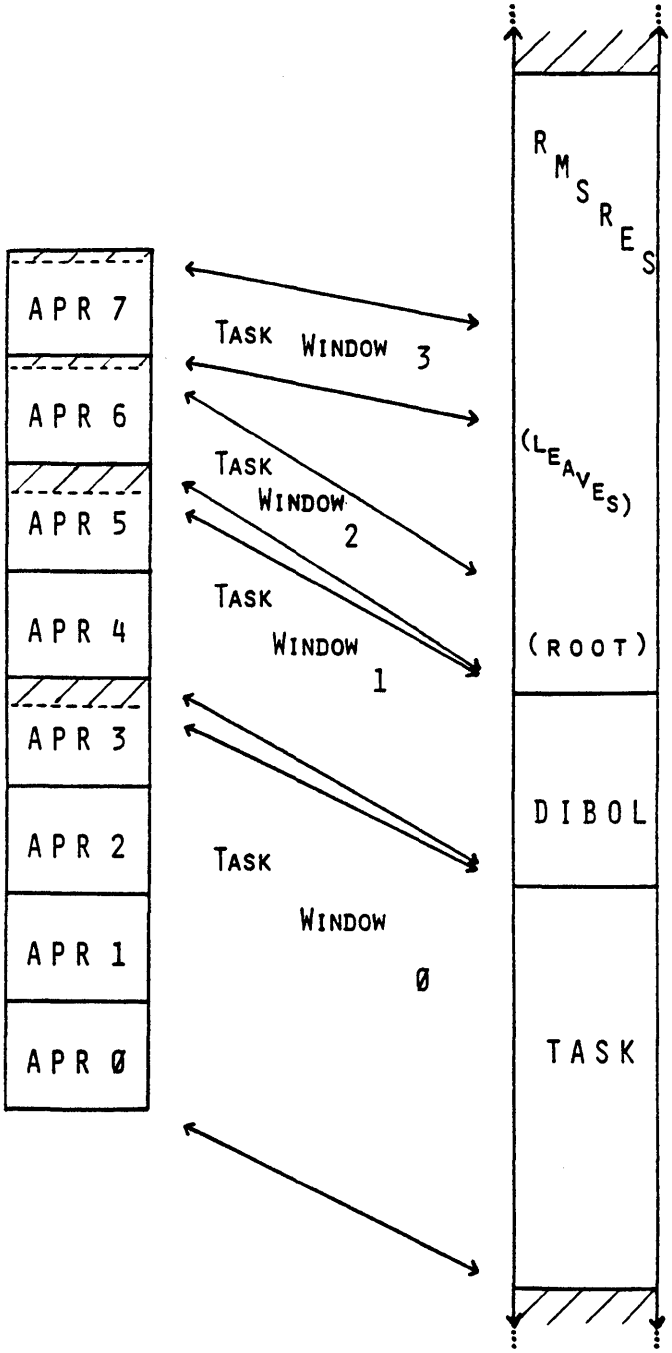


Figure B-4: Library With Memory-Resident Overlays

B.2.5 Clustered Libraries

As described in the previous section, multiple libraries are good for performance and potential sharing of physical memory, but not so good for task virtual address space, and minimum physical memory requirement (at least when only a single task is using the libraries). More flexibility is needed - and PDP-11 task structure provides it in the window mechanism. Windows are not statically associated with specific regions. They may be created and deleted as needed, at relatively low performance cost. In particular, when two libraries do not interact directly with one another (i.e., they do not need to be mapped simultaneously), they may alternately make use of the same task virtual address space (APR) range under control of the task's integral overlay run-time system.

As Figure B-5 illustrates, this structure presents the same task configuration as discussed in the previous section. Now, however, RMS and DIBOL share the same two APRs (6 and 7). At the moment, DIBOL is mapped by the task, and RMS is not. Because of this, RMS need not occupy physical memory (unless some other resident task is mapped to RMSRES), and the instantaneous physical memory requirement for the running task drops from 45 KW to 22 KW. If an RMS request occurs, of course, RMSRES must be mapped. However, DIBOL must first be unmapped, and hence again the instantaneous physical memory requirement is reduced (38 KW vs. 45 KW).

Thus, the task can run in as little as 38 KW of physical memory, though if there is less than 45 KW (appropriately distributed) RMSRES and DIBOL will "swap" against each other, resulting in disk I/O similar to overlaying. The effect is to use the physical memory controlled by the operating system as a code cache. When the supply is plentiful, performance approaches the optimal multiple library case. When memory gets tight, performance degrades gradually as disk I/O activity relieves the pressure (and improves again dynamically when more physical memory becomes available). Additional libraries (such as an FMS library) may be added to the cluster as long as the rule that all are independent of each other is followed. In such a case, the minimum physical memory requirement is unaffected unless one of the libraries added is larger than the previous largest library in the cluster.

P/OS and RSX-11M-PLUS systems allow the most dynamic use of memory, as described above. RSTS/E systems tie libraries to specific areas in physical memory, but allow use of this physical memory for other purposes when it is not required by the library that "owns" it. RSX-11M systems do not support demand region loading, and hence do not experience any reduction in required physical memory. All libraries must always be resident.

Also, the largest VIRTUAL address space requirement of any library in the cluster is all the user task sees. RMSRES, DAPRES (the library supporting RMS remote access via DECNET), FMS, and DIBOL can, for example, all share the same 8 KW (two APRs) of task virtual address space. This was, in fact, the original impetus for library clustering. The advantages in using physical memory were recognized later.

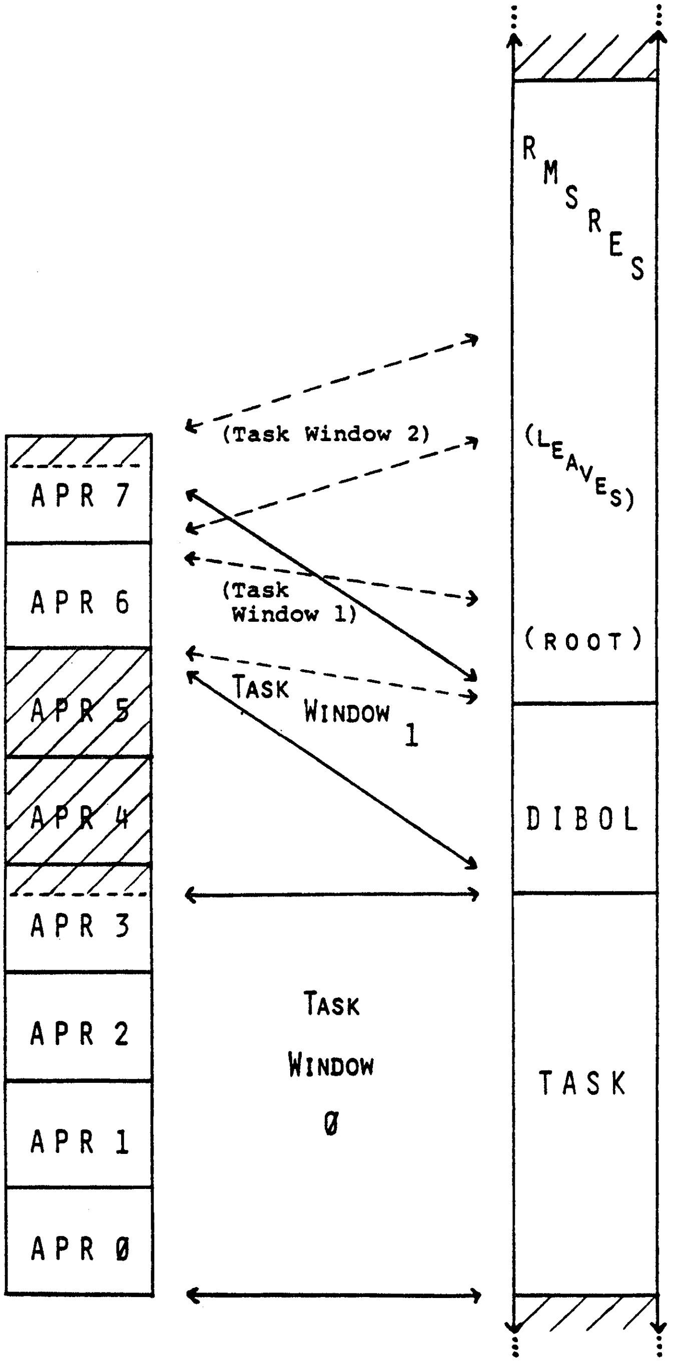


Figure B-5: Clustered Libraries

B.2.6 Cluster Libraries - Implementation Detail

The library clustering mechanism is implemented as an extension to existing Task Builder overlay structures. To the overlay run-time system, a library cluster resembles a null-rooted PLAS- overlaid co-tree, with each library a sub-tree. With the optional exception of the first library in the cluster, each of these sub- trees must itself have a null root. This structure simply aids the TKB in using its normal tree processing mechanisms to build the task structure.

When the first library in the cluster has a non-null root segment, TKB builds the task such that the loader will load and map this library root when the task itself is loaded. Thereafter, the task-resident overlay run-time system keeps watch over the co-tree mapping. If a reference into the library cluster causes its APR mapping to change from one library (region) to another, the mapping context for the displaced library (region) is saved on the stack and is restored upon exit from the newly-called library.

The first library in the cluster (assuming it has a non-null root) is always the initial library mapped. Therefore, any reference to another library in the cluster will always displace it, and it will always be re-mapped on exit. It is thus termed the "default" member of the cluster, for two reasons. First, mapping reverts to it when no other library has a routine in progress, and second, references made to it normally do not require any "push-down" mapping context to be saved on the stack - since the reference will not normally displace one of the other libraries' mapping. In other words, the default member of the cluster may usually be treated by the task just as if it were a non-clustered library. TKB takes advantage of this by suppressing generation of overlay run-time system autoload vectors for references to the non-null root of the default cluster member. The four words per root entry point saved by such direct addressing can be significant for default libraries with large numbers of entry points in the root - such as language OTS libraries. (If such libraries have PLAS-mapped "leaves" above the root, reference to entry points in the leaves will be autoload-vectored as usual).

If all libraries in the cluster have null roots, the first library to be called by the task takes the role of the default member of the cluster. Once it has been mapped, any reference to another cluster member will displace it, and restore its mapping on exit. Since it (as well as the other libraries) has a null root, autoload vectors will be generated for ALL entry points referenced by the task.

Note that this implementation does **not** require that all libraries in the cluster use the same number of APRs or the same number of virtual address windows. It **does** require, however, that they all be mapped, starting at the same task virtual address (APR boundary). This, in turn, means that all member libraries in a cluster must either be PIC, or built at the same virtual address.

B.2.7 Summary and Implications

Understanding how cluster libraries work makes it easier to understand (and remember) how to design and use them properly. To summarize:

- Library clustering allows tasks to use the same virtual address space range (APRs) for multiple purposes (leaving a larger range of virtual address space free for other use), while retaining the ease-of-development and code-sharability features of normal memory-resident libraries.

  There are two costs associated with use of cluster libraries:

  1. The overlay run-time system and its data structures (several hundred words total) must be built into the root of your task. If your task already required the overlay run-time system for other reasons, the additional size increase is relatively small.

  2. A reference to a non-default member of the cluster causes mapping/re-mapping operations to occur. The typical overhead is 2-10 milliseconds per reference (far less than a typical disk I/O operation), depending on processor speed and the number of windows in the affected libraries.

- Library clustering allows P/OS, RSX-11M-PLUS and RSTS/E operating systems to make more flexible use of physical memory, since a task maps only one library at a time. When memory is in short supply, a task may be able to run (albeit more slowly) which could not run at all if the libraries were not clustered. When a library is not mapped for long periods of time, the physical memory may be put to other use.

- Library clustering is **not** transparent to the libraries in the cluster. They must make no direct reference to each other (see below), and any which are not PIC must be built at the same starting virtual address.

Library clustering is not **fully** transparent to the user task:

1. The nature of the "push-down" mapping mechanism requires that any CALL into the cluster be of the form JSR PC,... (rather than, for example, JSR R5,...). While JMPs into the cluster are acceptable, co-routine linkages are unacceptable.

2. In general, the task must access the cluster only as described above. The task must not "remember" virtual addresses within the cluster, and cannot use the .NAME facility to load library data tables.

3. Since push-down mapping affects the stack depth, parameters may be passed on the stack across the task/cluster interface only if an argument pointer is used.

The above three restrictions are relaxed only in the case of the default member of the cluster, and only then when it is known that no other member is currently "pushed down" on the stack (see next section).

B.2.8 Inter-Library References and Miscellaneous Points

That completes this description of how clustering works and what its limitations are. One of the limitations clearly needs a work-around. For example, a language or FMS library needs access to the FCS or RMS library for file access, and to say that such combinations cannot be clustered is just not acceptable.

Any work-around is not a part of the clustering mechanism per se. It is a cooperative effort on the part of the interacting libraries. FCS and RMS, for example, both provide linkage routines (modules FCSVEC and R0RMSC, respectively) which must be built into any library whose code contains direct references to the FCS/RMS functions (e.g., OPEN$/$OPEN). These special modules field any FCS/RMS request made within the library and pass it - using the low-core absolute linkage to the FCS or RMS task-resident impure area - to another FCS or RMS routine which exists in the task itself. This routine then calls into the FCS/RMS library, thus avoiding any direct inter-library call. (In the case of FCS, the "routine" is simply a JMP through an autoload vector.)

Note, however, that the FCS/RMS global entry point symbols now appear in the calling library's symbol table. They will be visible to the user task, unless the calling library is built with .GBLXCL statements for each such symbol. Since it is not normally desirable to "vector" RMS references from the task through some intermediate library, the .GBLXCL mechanism is generally appropriate. In the case of FCS, it is mandatory, as these same symbols are used as the entry points to the FCS library and will conflict if not suppressed.

The mechanism described above is equally useful for non-clustered libraries which need access to FCS or RMS without being bound to specific external routine entry points. This is independent of whether the FCS/ RMS code is task-resident or in a library of its own. An alternative approach is for the calling library to invoke a service routine of its own in the task, which then dispatches to FCS/RMS directly.

Finally, the calling library must contain **nothing** (data structures, file specification strings, etc.) that the FCS/RMS library needs to "see" in the execution of its duties. This is the caller's end of the bargain. RMS completion routines are called from the in-task RMS linkage code after exit from RMSRES, and hence may be present in the caller's library (which has by then been re-mapped). "Get space" (GSA) routines provided by the user are called with the RMS library mapped, but will execute correctly regardless of location **provided** the address given to RMS reflects an in-task autoload vector (the RMS mapping will be "pushed down" if necessary to map the GSA routine, and will be restored transparently on GSA routine exit).

This last situation reflects one of the potential pitfalls of "default" libraries. If such a library is clustered with RMS and provides a GSA routine address in its (non-null) root segment, when an RMS operation is performed, the call out of RMSRES to the GSA routine will, in fact, transfer control to some random location within the RMS library. This is because the default member is not mapped at the time and the overlay run-time system is not invoked on the reference (no autoload vector was generated). There are three possible remedies:

1. Place the GSA routine in the task rather than in the library.
2. Place the GSA routine in a PLAS-overlaid "leaf" of the default library - whose entry points are autoload-vectored.
3. Build the first library as the other cluster members with a null root - which will force autoload vectoring throughout.

A situation similar to the above applies to synchronous traps. Any trap service routine in the cluster must be suitably autoload-vectored through the controlling task, as the routine may not be mapped at the moment the trap occurs. At present, the overlay run-time system's handling of its data structures is not suited to **any** use of overlaying during asynchronous trap processing unless it is the **only** time overlaying occurs. An enhancement is under development, and when it appears, asynchronous trap servicing may be handled as described for synchronous trap servicing.

B.2.9 WRITE Access to Clustered Libraries

For P/OS V1.7, the overlay run-time system has been modified to allow write access to (non-"default") clustered libraries that have not been installed read-only.

Such libraries may be useful for the following purposes:

- Passing information between cooperating application tasks (the tasks should provide their own access synchronization)
- To extend the effective available read/write virtual memory usable for task impure data

In both cases, the task(s) must ensure that the library is mapped by calling a routine in the library that does not RETURN to the caller until any access to the read/write data is completed. This is not normally necessary for a non-null-rooted 'default' cluster member, since it is usually already mapped as desired.

B.2.10 NULLIB

The special non-null-rooted default cluster member NULLIB was provided in previous P/OS releases for two purposes:

- to guard against potential memory fragmentation problems that might cause task deadlock in certain instances.
- to provide better performance in cases when a null-rooted cluster member would otherwise become the 'effective' default member of the cluster and would be re-mapped (and potentially re-loaded from disk) unnecessarily. Assuming that, typically, the application would access other cluster members than the first one accessed, re-mapping that first member after every access to some other one could prove costly.

The increased physical memory now standard for P/OS makes the memory fragmentation problems considerably less likely for most applications. Also now that the RMSRES and POSSUM resident libraries are fixed in memory, there is no possibility of disk loading overhead when one of these becomes the effective default member of a cluster.

As a general rule, when all members of a library cluster have null roots, the application should attempt to ensure that the first library accessed (which will become the effective default cluster member) is the one that the application will refer to most frequently. This will minimize the likelihood of unnecessary mapping and possible disk loading.

B.3 OPTIONS IN TASK ORGANIZATION

The following examples have been chosen to demonstrate the options for organizing a task which needs to use the following P/OS services: RMS, Callable System Services (POSSUM), and User Interface Services (POSRES).

- Option 1: FLAT.TSK

  Completely flat task structure. All task modules are in a single segment. The three resident libraries are mapped in their own APRs.

  Advantages: Easiest taskbuild command file to construct. No special attention has to be made to placement of data. Less mapping activity necessary.

  Disadvantages: Most costly in terms of virtual memory space.

- Option 2: CLUST.TSK

  Task region is a flat structure. The resident libraries are clustered against each other.

  Advantages: Because the resident libraries share APR mapping, other APRs are freed for task use.

  Disadvantages: More mapping and unmapping required by runtime services.

- Option 3: VECTOR.TSK

  Task space contains minimal code. Most code has been relocated to a user-written cluster library (USRRES). This library is clustered against RMSRES, POSSUM, and POSRES. Data required by USRRES must reside in task address space.

  Advantages: Even more address space available in task area.

  Disadvantages: More difficult to organize task code and data. A vectoring scheme must be implemented for library data references and subroutine calls from one library in the cluster to another. Extra mapping and unmapping by the overlay runtime system.

- Option 4: OVERLAY.TSK

  Task itself is overlaid. All code and data which are needed to call RMSRES, POSSUM, and POSRES reside in one branch of the tree. Code not using these libraries is overlaid against code using libraries.

Advantages: Extends task virtual address space considerably because code/data for library references are removed from root segment of task and are consequently able to be unmapped.

Disadvantages: More complicated .ODL file. Requires some knowledge of library-specific requirements. Requires logical separation of functionality.

Implementation details follow.

B.3.1 FLAT.TSK

This is the simplest structure to concoct. In a flat task structure, task control can be directly transferred to mainline code. Nevertheless, the example uses the control module ROOT in order to provide a basis for comparison with the alternative options.

B.3.1.1 FLATBLD.CMD -

```
;           control  mainline  RMS,POSSUM
;           module   code      & POSRES
;                                  data
FLAT, FLAT = ROOT, MAINLINE, ROOTDATA
/

; each library will have its own apr assignment
LIBR = POSRES:RO
LIBR = RMSRES:RO
LIBR = POSSUM:RO

TASK = FLAT

; define and assign logical units needed by this task:
UNITS = 7
ASG = SY:1:2:3:4:7
ASG = TI:5
GBLDEF = TRGTLN:1         !lun for target file (call to PROATR)
GBLDEF = HL$LUN:2         !lun for help file (not used in this example)
GBLDEF = MN$LUN:3         !lun for menu file
GBLDEF = MS$LUN:4         !lun for message file (not used in this example)
GBLDEF = TT$LUN:5         !lun for POSRES services terminal I/O
GBLDEF = WC$LUN:7         !lun for POSRES wildcard directory search
GBLDEF = TT$EFN:2         !event flag for POSRES terminal I/O

EXTSCT = FL$BUF:2000      !provide a buffer for file specs (OLDFIL)
EXTSCT = MM$BUF:2000      !buffer for multi-choice menu (OLDFIL)
EXTSCT = MN$BUF:2000      !buffer for additional options menu (OLDFIL)
//
```

B.3.1.2 ROOT.MAC -

```
     .title    coderoot
     .ident    /01.00/

     .enabl    lc
     .mcall    exit$s
;+
; The operational code for this test task is found in MAINLINE.MAC.
; Necessary data structures are in ROOTDATA.MAC.
```

```
;-
     .sbttl    Task root code
     .psect
START:
     MOV       #DATATBL,R0         ;table of subroutine parameter bloc
     CALL      MAINLN              ;mainline task code
     EXIT$S                        ;done

     .sbttl    Data table to drive mainline code
     .psect    roodat,d,rw,con
DATATBL:
     .WORD     OLFPAR              ;call oldfil first
     .WORD     FAB                 ;will need FAB address to set up
                               ;call to rms
     .WORD     LUN
     .WORD     FIDBUF              ;three-word buffer for file id
     .WORD     POSPAR              ;possum (proatr) parameter block)

     .END      START
```

B.3.1.3 MAINLINE.MAC -

```
     .title    Mainline code
     .ident    /01.00/
     .enabl    lc

     .mcall    qiow$s, alun$s
     .mcall    $parse, $search, $compare, $fetch, $store
     .mcall    fhdof$
     fhdof$                        ;define file header offsets for
                               ;UC.CON


;+
; Mainline code for test task.
; This module will:
;    1.  Establish proper lun assignments for terminal and target file
;    2.  Call OLDFIL in POSRES to allow choice of target file.
;    3.  Call RMS to determine FILE-ID of target file.
;    4.  Call PROATR in POSSUM to determine whether file is contiguous
;    5.  Output result to screen.
;
; Most data for this module lives in ROOTDATA.MAC.  This will permit
; this module to be incorporated into a resident library which may
; cluster against RMSRES, POSSUM, and POSRES and to share data with
; that library.
;
; There is some read-only data in this module which is included to
; demonstrate its possible use in a cluster library.
;
```

```
; Inputs:       R0 -> Data table in the format:
;                       .word     oldfil parameter block
;                       .word     address of rms FAB
;                       .word     address of lun for target file
;                       .word     address of 3-word buffer for file id
;                       .word     address of parameter block for possum
;
; Outputs:      none
;
;-
     .sbttl     Mainline code
     .psect
MAINLN::
     MOV        (R0)+,R5  ;parameter block for oldfil
     MOV        R0,-(SP)  ;preserve input pointer across call
     MOV        R5,-(SP)  ;and also oldfil parameter block address
     CALL       OLDFIL              ;get a file spec
     MOV        (SP)+,R5  ;restore oldfil parameter block
     MOV        (SP)+,R0  ;restore input table
     TST        @2(R5)              ;successful?
     BLE        10$                 ;no

     MOV        (R0)+,R2  ;get fab address
     $STORE     @8.(R5),FNS,R2      ;store size of filespec (from oldfil)
     MOV        @(RO)+,R1
     $STORE     R1,LCH,R2 ;enter lun in FAB

     $PARSE     R2                  ;call RMS
     $COMPARE #SU$SUC,STS,R2        ;success?
     BNE        10$                 ;no

     $SEARCH    R2                  ;get file id into nam block
     $COMPARE #SU$SUC,STS,R2        ;successful?
     BNE        10$                 ;no, error

     $FETCH     R1,NAM,R2       ;get NAM block address
     MOV        (R0)+,R3        ;buffer to receive file id
     $FETCH     0(R3),FID,R1        ;retrieve File ID into buffer

     MOV        (R0)+,R5  ;proatr parameter block address
;
; The proatr parameter looks as follows:
;    .word      5
;    .word      addr. of 8-word status block
;    .word      addr. of request word
;    .word      addr. of buffer to construct attribute list
;    .word      addr. of file-id buffer
;    .word      lun addr.
;
     MOV        #0,@4(R5)       ;initialize request - get file attributes
                                    ;   by file id
```

```
     MOV       6(R5),R0        ;get address of attribute buffer
; fill in attribute list:
     MOVB      #3,(R0)+  ;attribute type is file characteristics
     MOVB      #2,(R0)+  ;two words
     MOV       R0,(R0)   ;construct a buffer address
     ADD       #4,(R0)+  ;point past attribute list
     CLR       (R0)+           ;end of attribute list
     CLR       (R0)            ;initialize cell to receive
                                ;characteristics
;
;    .BYTE     3,2             ;attribute type,size
;    .WORD     1$              ;location of output
;    .WORD     0               ;null ends attribute list
; 1$:          .WORD     0               ;returned value from proatr
;

     CALL      PROATR              ;get attributes
     MOV       2(R5),R1        ;status block address
     TST       2(R1)               ;success?
     BLE       10$                 ;no, pass back error

     BIC       #^C<UC.CON>,(R0)    ;clear all but contiguous bit in
                                   ;   attribute word
                                   ;if bit set, then file contiguous
                                   ;if bit clear, then file
                                ;   not-contiguous
                                   ;   and (R0)=0


;pick up address of messages
; code must be pic since this will go into a /PI library
     MOV       PC,R1               ;assume contiguous
     ADD       #CONTIG-.,R1
     MOV       #CONTIZ,R2
     TST       (R0)                ;contiguous if non-zero
     BNE       5$                  ;yes, contiguous
     MOV       PC,R1               ;non-contiguous
     ADD       #NOCONT-.,R1
     MOV       #NOCONZ,R2
5$:
     ALUN$S    #5,#"TI         ;point terminal lun to the right place
     CLR       -(SP)               ;create io status block
     CLR       -(SP)
     MOV       SP,R3               ;point to it
     QIOW$S    #IO.WVB,#5,#1,,R3,,<R1,R2,#40> ;entertain
     MOV       (SP)+,(SP)+         ;clear off iosb
10$:
     RETURN

     .sbttl    Miscellaneous data
     .psect    misdat,d,ro
```

```
contig:         .ascii    <33>/[2J/<33>/[10;10H/
     .ascii    /The file you have chosen is contiguous/
contiz = .-contig

nocont:         .ascii    <33>/[2J/<33>/[10;10H/
     .ascii    /The file you have chosen is not contiguous/
noconz = .-nocont
     .even

     .end
```

B.3.1.4 ROOTDATA.MAC -

```
     .TITLE    DATA FOR TASK ROOT
     .ident    /01.00/
     .enabl    lc

     .mcall    fab$b, nam$b, pool$b

;+
; This module contains data needed by resident libraries (and in some
; cases in-task root).  Data must be mapped when resident library is
; called.
;-
     .psect    roodat,d,rw,con

     .SBTTL    DATA FOR OLDFIL

prompt:         .ascii    /Step right up!  See if file is contiguous/
promz = .-prompt
     .even
prompz:         .word     promz                ;indirect reference for
                                 ;oldfil

olfpar::
     .word     olfpaz                 ;size of parameter block
     .word     olfsts                 ;status block address
     .word     numcho                 ;number of choices
     .word     fna                    ;file name string
     .word     ofsiz                  ;array (1 element) for
                                 ;filespec size(s)
     .word     nowild                 ;no wildcard default
     .word     nosiz                  ;no length for no wildcard default
     .word     prompt                 ;text
     .word     prompz
     .word     nomsg                  ;no message
     .word     nosiz
     .word     nomsg
     .word     nosiz
olfpaz = <.-olfpar>/2 - 1
```

```
olfsts:         .blkw       2           ;status block
numcho:         .word       1           ;one choice only
ofsiz:          .blkw       1           ;will contain size of filespec
nowild:                                 ;no wildcard
nomsg:                                  ;no message
nosiz:          .word       0           ;no size


     .SBTTL     DATA FOR RMS CALLS

lun::           .word       trgtln      ;target lun

fidbuf::        .blkw       3


     .SBTTL     DATA FOR POSSUM CALL

pospar::
     .word      5
     .word      stablk      ;addr. of 8-word status block
     .word      buf1        ;addr. of request word
     .word      buf2        ;addr. of buffer to construct call
     .word      fidbuf      ;addr. of buffer containing file-id
     .word      lun         ;addr. of lun for file

stablk:         .blkw       8.
buf1:           .word       0           ;request word
buf2:           .blkw       8.          ;enough for an attribute


     .SBTTL     RMS DATA STRUCTURES

fna: .blkb      256.        ;buffer for file specification

dna: .ascii     /SY:;0/     ;look for the latest version on default volume
dns = .-dna
     .even                  ;force even boundary
dnasiz:         .word       dns         ;indirect reference for oldfil

rss =           255.
rsa: .blkb      rss         ;resultant string

ess =           255.
esa: .blkb      ess         ;expanded string
     .even

fab::           fab$b
     f$nam      nam         ;nam block address
     f$fna      fna         ;file specification address
     f$dna      dna         ;default file spec. addr.
     f$dns      dns         ;dna size
```

```
     fab$e

nam: nam$b
     n$esa      esa        ;expanded string address
     n$ess      ess        ;size of esa buffer
     n$rsa      rsa        ;resultant string address
     n$rss      rss        ;size of rsa buffer
     nam$e

     pool$b
     p$fab      1          ;one fab
     p$buf      512.       ;one-block buffer
     p$bdb      1          ;one buffer descriptor block
     pool$e

     .end
```

B.3.2 CLUST.TSK

This task makes use of the cluster library facility to free some APRs which would otherwise be used to map to resident libraries. The task itself is still a flat structure. Because the cluster library facility uses the overlay runtime system the task incorporates an overlay descriptor file. The .ODL file describes this flat structure in a manner similar to the specification of modules in the previous example (FLAT.TSK).

Only one member of a library cluster may have a non-null root. If there is a library which meets this description, it must be the default library of the cluster (appear first in the CLSTR option line). Mapping to this default library is always restored after the completion of a call to another member of the cluster. Normally a higher-level language OTS falls into this category.

In the case in which there is no library with a non-null root, the library which is INVOKED first becomes the default library de facto, regardless of its position in the CLSTR option line. The consequence of this is that if the task makes repeated calls to a particular library, it would be advantageous to make a call to the most often invoked library before calling any less frequently used library.

B.3.2.1 CLUSTBLD.CMD -

```
CLUST, CLUST = CLUST/MP

CLSTR = RMSRES,POSRES,POSSUM:RO

TASK = CLUST

UNITS = 7
ASG = SY:1:2:3:4:7
ASG = TI:5
GBLDEF = TRGTLN:1
GBLDEF = HL$LUN:2
GBLDEF = MN$LUN:3
GBLDEF = MS$LUN:4
GBLDEF = TT$EFN:2
GBLDEF = TT$LUN:5
GBLDEF = WC$LUN:7

EXTSCT = FL$BUF:2000
EXTSCT = MN$BUF:2000
EXTSCT = MM$BUF:2000
//
```

B.3.2.2 CLUST.ODL -

```
;              control
;              module
     .ROOT     ROOT- OTHER- RMSROT

;              mainline RMS,POSSUM
;                code    & POSRES
;                           data
OTHER:         .FCTR     MAINLINE- ROOTDATA

; include RMS root modules (RMSROT) so that POSRES routines
;   can call RMS (see discussion of USRRES, below)
@LB:[1,5]RMSRLX

     .END
```

B.3.3 VECTOR.TSK and USRRES.TSK

This example demonstrates

1. How to remove code from task space and place it in a resident library.
2. How to build a resident library with a null root so that the library may be used as part of a cluster of resident libraries.
3. How to address data in the task root from the resident library.
4. How to provide a facility to call other members of the cluster from the user-written cluster library.

In this example, the user-written library is intended to be installed read-only and the library cluster mapped RO. As a result, there cannot be any read/write data in the library; there is, however, some read-only data in this library for demonstration purposes.

Note

> In the following discussion, the terms "task space", "task root", and "task" all refer to a task which maps to the resident library USRRES.

B.3.3.1 USRRESBLD.CMD -

```
; Build a PIC
; resident library
;                         Include .STB
;                         file for
;                           tasks which
;                         map to this
;                         library
;
USRRES/-HD/PI/LI, USRRES, USRRES   =   USRRES/MP

PAR = USRRES:0:0
STACK = 0
;
; Insure that the symbol table for this library will contain
; references to the externally available entry point(s):
;
GBLREF = MAINLN
;
; Force the task builder to check that the root vectoring module
```

```
; is included in the task using this library for cross-cluster calls
; and accessing data in the root.
;
GBLINC = YANKME
;
; This library calls POSRES and POSSUM services (OLDFIL and
; PROATR, respectively).  The entry point names must be resolved
; when building this library, but cannot be included in the library
; .STB, since references to these symbols from the task must be
; resolved by the library which contains the routine. (See discussion
; of cross-cluster calls.)
;
GBLXCL = PROATR
GBLXCL = OLDFIL
//
```

B.3.3.2 USRRES.ODL - Resident Library - The following .ODL file describes the resident library. It is a non-overlaid resident library with a null root. Any module without code or data allocation will suffice as a root; SYSLIB includes the module NULL for this purpose.

```
     .ROOT     LB:[1,1]SYSLIB/LB:NULL- !(CODE)

CODE:          .FCTR     *MAINLINE- RESVEC- SYSLIB/LB:R0RMSC:RMSSYM

     .END
```

B.3.3.3 Discussion - The module MAINLINE is declared autoloadable, indicating to the task builder to mark its global entry point(s) appropriately in the symbol table of USRRES. Calls to these routines from the task will generate autoload vectors. The autoload indicator is used in conjunction with the GBLREF option in the taskbuild command file. The GBLREF option instructs the taskbuilder to place the symbol in the .STB file for this library. This permits the calling task to have access to that entry point.

The modules R0RMSC and RMSSYM from SYSLIB permit this resident library to call RMSRES, which may be clustered against USRRES. This is accomplished by vectoring these calls through the task root. The vectoring module RESVEC illustrates the use of this scheme for those components which do not already provide a similar facility (see below).

Vectoring is required for two purposes:

1. Accessing data in the task root from the library.
2. Cross-cluster calls.

Since the library must be taskbuilt before the referencing task, the library cannot reference the task directly. The task may pass data addresses, for example, as parameters to library routines. Alternatively, the library may address task space via a limited number of fixed addresses which remain meanings from task to task. This area is referred to as the "low-core" context of the task image.

As part of the low-core context, the taskbuilder automatically deposits the address of the first word of the psect $$VEX1 in the location $VEXT. Since $VEXT is always at the same address in all tasks, a library may use this location to find the beginning of the $$VEX1 psect, whose actual virtual location will undoubtedly vary from task to task.

Negative offsets from the beginning of the $$VEX1 PSECT are reserved for Digital impure area vectors. Users may employ any positive offset; however, care should be taken to guarantee that other applications (e.g. high-level languages) are not already using the locations in $$VEX1.

The psect has the following attributes:

```
        .PSECT $$VEX1,D,GBL,REL,OVR
```

The OVR attribute allows contributions to the PSECT to come from different sources while maintaining a fixed point of reference from the beginning of the PSECT. A SYSLIB module contains a global symbol pointing to the beginning of this PSECT. The address of this symbol is deposited by the taskbuilder in the location $VEXT.

Therefore a library routine may execute the following instruction to point Rn to the beginning of $$VEX1.

```
        MOV     @#$VEXT,Rn
```

Rn is any one of the general purpose Registers. The **absolute** addressing mode (@) is required to guarantee position independence of the library.

The library routine RESVEC makes use of the fact that a task built against this library will contain impure area vectors required by USRRES. The resident library can insure that the vector-declaring module is included in the task by using the GBLINC option; this option results in an error in a task built against this library if the specified symbol is not resolved.

A module in USRRES callS PROATR by a normal JSR, PC:

```
        CALL   PROATR
```

A module must be included in the resident library to vector this call through the task root.

B.3.3.4 Excerpt from RESVEC.MAC -

```
; This global symbol must be excluded from the .STB of USRRES.
;   Otherwise, the taskbuilder would not be able to resolve the symbol
;   unambiguously between USRRES and POSSUM.  The GBLXCL option in the
;   USRRES taskbuild command line instructs the taskbuilder to exclude
;   the symbol.
PROATR::
; This code is written in this manner so as to preserve all registers.
     MOV     @#$VEXT,-(SP) ;get pointer to $$VEX1 psect
                           ; (SP) address of MYVECT (see below)
     MOV     @(SP)+,-(SP)  ;point to table
                           ; (SP) addr of INDIRECT
     ADD     #2,(SP)       ;PROATR transfer point is second in table
                           ; (SP) addr. of 2nd word in table
     MOV     @(SP)+,-(SP)  ;get contents of 2nd word of table
                           ; (SP) addr. of "PROATR"
; In actuality, the stack contains the address of an overlay runtime
;   autoload vector.
     JMP     @(SP)+        ;transfer to "PROATR" (overlay runtime
                           ;system)
```

At this point, the task is executing in the overlay runtime system, which will save the mapping context of USRRES, remap to POSSUM, and transfer control to PROATR. At the conclusion of PROATR, the overlay runtime system will restore mapping to USRRES and return control to the instruction following the call to PROATR.

A similar procedure would be followed to address the data space in the task root. Here a register is available.

```
     MOV     @#$VEXT,R0    ;address of MYVECT
     MOV     (R0),R0       ;point to INDIRECT (vector table)
     MOV     4(R0),R0      ;contents of entry is address of data
                           ;  table
```

B.3.3.5 VECTOR.MAC (Root vector module) -

```
; Define symbol required by USRRES
YANKME == 0

     .PSECT $$VEX1,D,GBL,REL,OVR
```

```
; This label will be equivalent to the value which the taskbuilder has
;  deposited in $VEXT because the psect $$VEX1 is OVR (overlaid).
MYVECT:

; It is good practice to add a level of indirection.  This will insure
;   that this use of this particular psect will consume only a single
;   word.
     .WORD  INDIRECT

     .PSECT JMPTBL,D,RO,CON
INDIRECT:
     .WORD  OLDFIL         ;used in cross-cluster calls
     .WORD  PROATR         ;used in cross-cluster calls
     .WORD  PNTDAT         ;used to point to data in task root

     .PSECT IMPURE,D,RW,CON
PNTDAT:
     .BLKB  120.           ;read/write data area

     .END
```

B.3.3.6 VECTORBLD.CMD -

```
; Command file to build task which maps to user-written resident
; cluster library
VECTOR, VECTOR = VECTOR/MP

CLSTR = RMSRES,POSRES,POSSUM,USRRES:RO
TASK = VECTOR

UNITS = 7
ASG = SY:1:2:3:4:7
ASG = TI:5
GBLDEF = TRGTLN:1
GBLDEF = HL$LUN:2
GBLDEF = MN$LUN:3
GBLDEF = MS$LUN:4
GBLDEF = TT$EFN:2
GBLDEF = TT$LUN:5
GBLDEF = WC$LUN:7

EXTSCT = FL$BUF:2000
EXTSCT = MN$BUF:2000
EXTSCT = MM$BUF:2000
//
```

B.3.3.7 VECTOR.ODL -

```
     .ROOT  ROOT- OTHER- RMSROT

;                          task root
;                          vectoring
;                           module
OTHER:      .FCTR  ROOTDATA- VECTOR

@LB:[1,5]RMSRLX

     .END
```

B.3.4 OVERLAY.TSK

This example shows that a program can separate functionality in such a way that the data required by a resident clustered library need not always be mapped.

In order to achieve this goal, one must first carefully analyze the potential for functional independence of routines. The example is structured to demonstrate a group of routines which call services in RMSRES, POSSUM, and POSRES and a second group of routines which do some computation and QIO's to the terminal.

The second stage in the process is to determine which modules are required by the resident library in question, and then force the taskbuilder (through the .ODL file) to place these modules in the appropriate branch of the overlay tree structure. If the modules were not specified by the .ODL file, they would be in the task root rather than in a branch and therefore would consume shared virtual address space and be mapped even when not needed.

B.3.4.1 OVERLAY.ODL -

```
; The root module has changed from previous examples because of the
;   added calls (to COMPUTE and WTRES)
     .ROOT  ROOT2- (LEFT, RIGHT)

; The module CLLWTR is included to call WTRES (wait for resume
; key) in POSRES.  The module COMPUTE displays a message to press
; <RESUME> when the user is ready to continue.  In order to keep
; all references to POSRES in the "left" branch of the tree, the
; .ODL forces the root to transfer control up-tree in order to
; call POSRES.
LEFT:       .FCTR  *MAINLINE- *CLLWTR- ROOTDATA- BUFS- RMSROT- LIB
; Force other syslib references into this branch
LIB: .FCTR  LB:[1,5]SYSLIB/DL

; These are the buffers required by POSRES services in this example
BUFS:       .FCTR  SYSLIB/LB:PTIMP:PTFLF:FLFAB:PTDUM

RIGHT:      .FCTR  *COMPUTE- SYSLIB/LB:EDTMG

@LB:[1,5]RMSRLX

     .END
```

B.3.4.2 OVERLABLD.CMD -

```
OVERLAY, OVERLAY = OVERLAY/MP

CLSTR = RMSRES,POSRES,POSSUM:RO
TASK = OVERLY

UNITS = 7
ASG = SY:1:2:3:4:7
ASG = TI:5
GBLDEF = TRGTLN:1
GBLDEF = HL$LUN:2
GBLDEF = MN$LUN:3
GBLDEF = MS$LUN:4
GBLDEF = TT$EFN:2
GBLDEF = TT$LUN:5
GBLDEF = WC$LUN:7

EXTSCT = FL$BUF:2000
EXTSCT = MN$BUF:2000
EXTSCT = MM$BUF:2000
//
```

B.3.4.3 ROOT2.MAC -

```
     .TITLE ROOT2
     .ident /01.00/

     .enabl lc
     .mcall exit$s
;+
; The operational code for this test task is found in MAINLINE.MAC
; (Section B.3.1.3).
; Necessary data structures are in ROOTDATA.MAC.
; (Section B.3.1.4).
;
; The alternate mapping is COMPUTE.MAC.  All data for that segment
; is in the same module.
;-

     .SBTTL TASK ROOT CODE
     .psect
START:
     MOV    #DATATBL,R0    ;table of subroutine parameter blocks
     CALL   MAINLN         ;mainline task code - "LEFT"

     CALL   COMPUTE        ;do something else - "RIGHT"

     CALL   CLLWTR         ;wait for the resume key - "LEFT"
                           ;  do the waiting in the posres services
                           ;  branch of the tree
```

```
     MOV     #DATATBL,R0     ;just for fun, call first operation again
     CALL    MAINLN          ;
     EXIT$S                  ;done

     .SBTTL DATA TABLE TO DRIVE MAINLINE CODE
     .psect roodat,d,rw,con
DATATBL:
     .WORD   OLFPAR          ;call oldfil first
     .WORD   FAB             ;will need FAB address to set up call to rms
     .WORD   LUN
     .WORD   FIDBUF          ;three-word buffer for file id
     .WORD   POSPAR          ;possum (proatr) parameter block)

     .END    START
```

B.3.4.4 CLLWTR.MAC -

```
     .TITLE CLLWTR - CALL WTRES FROM OVERLAY
     .ident /01.00/

;+
; This module will call the POSRES service WTRES (Wait for Resume key).
; It is placed in an overlay branch in order to segregate all posres
; calls from the root and/or other branches, i.e. to localize virtual
; address requirements for calling posres services.
;-

CLLWTR::
     CALL    WTRES
     RETURN

     .END
```

B.3.4.5 COMPUTE.MAC -

```
     .TITLE COMPUTE
     .ident /01.00/

     .enabl lc
     .mcall qiow$s, mrkt$s, wtse$s
;+
; This module will consume space and time.  The overlay branch which
; shares this virtual address space will do more interesting things,
; e.g. call RMS, POSSUM, and POSRES.
; After this module does its thing, it will instruct the observer to
; press the resume key.  Since all calls to POSRES are in the other
; branch, this will return to ROOT2 to allow the root to transfer
; control to the POSRES-calling overlay.
;-
```

```
MRKEF = 3                 ;event flag to mark time

     .psect
COMPUTE::
     MRKT$S #MRKEF,#2,#2  ;mark time for 2 seconds

     MOV    #DATBUF,R0    ;address of data buffer
     MOV    #DATBUZ,R1    ;size in words
     MOV    #1,R2         ;fill buffer with miscellaneous garbage
1$:
     MOV    R2,(R0)+      ;enter first word
     INC    R2            ;bump data
     SOB    R1,1$
2$:
     MOV    #DATBUF,R0    ;now add it up
     MOV    #DATBUZ,R1
     CLR    R2            ;double precision
     CLR    R3
3$:
     ADD    (R0)+,R3
     ADC    R2
     SOB    R1,3$
4$:
     MOV    R2,DPHIGH     ;copy to memory
     MOV    R3,DPLOW
5$:
     MOV    #OUTBUF,R0    ;set up for $edmsg
     MOV    #FORMAT,R1    ;input string
     MOV    #ARGBLK,R2    ;argument block
     CALL   $EDMSG        ;format output
10$:
     WTSE$S #MRKEF        ;let user read previous message
15$:
     QIOW$S #IO.WVB,#5,#1,,#IOSB,,<#OUTBUF,R1,#40> ;print new one
20$:
     RETURN

     .SBTTL DATA FOR THIS BORING ROUTINE
     .psect boring,d,rw

DATBUZ = 100.                    ;a hundred bottles of beer on the wall
DATBUF:     .BLKW  DATBUZ

; KEEP THESE TWO VALUES TOGETHER, IN THIS ORDER:
DPHIGH:     .WORD  0
DPLOW:      .WORD  0

OUTBUF:     .BLKB  200.          ;buffer for output message

     .NLIST BEX
FORMAT:     .ASCII <33>/[2J/<33>/[10;10H/ ;clear screen and
```

```
                                                ;locate cursor
     .ASCII /This is the total:  %T.  Whoopee!/
     .ASCIZ /Press <RESUME> to continue./
     .EVEN
     .LIST  BEX

ARGBLK:     .WORD  DPHIGH
     .WORD  0

IOSB:       .BLKW  2

     .END
```

APPENDIX C

FILES-11 ON-DISK STRUCTURE SPECIFICATION

This appendix provides a specification of the on-media structure that is used by Files-11. Files-11 is a general purpose file structure which is intended to be the standard file structure for all medium to large PDP-11 systems. Small systems such as RT-11 have been specifically excluded because the complexity of Files-11 would impose too great a burden on their simplicity and small size.

This document describes structure level 1 of Files-11, also called ODS-1 (On-Disk Structure Version 1). This has been implemented on P/OS and on RSX. This document describes the final level of functionality for ODS-1. Structure level 2 (ODS-2) has been implemented on VMS and is the basis for all new disk structure enhancements.

C.1 MEDIUM

Files-11 is a structure which is imposed on a medium. That medium must have certain properties, which are described in the following section. Generally speaking, block addressable storage devices such as disks and Dectape are suitable for Files-11. Therefore, Files-11 structured media are generically disks.

C.1.1 Volume

The basic medium that carries a Files-11 structure is a volume (also often called a unit), and is defined as an ordered set of logical blocks. A logical block is an array of 512 8-bit bytes. The logical blocks in a volume are consecutively numbered from 0 to n-1, where the volume contains n logical blocks. The number assigned to a logical block is called its logical block number, or LBN. Files-11 is theoretically capable of describing volumes up to 2^32 blocks in size. In practice, a volume should be at least 100 blocks in size to be useful. Current implementations of Files-11 will handle volumes up to 2^24 blocks.

The logical blocks of a volume must be randomly addressable. The volume must also allow transfers of any length up to 65k bytes, in multiples of four bytes. When a transfer is longer than 512 bytes, consecutively numbered logical blocks are transfered until the byte count is satisfied. In other words, the volume can be viewed as a partitioned array of bytes. It must allow reads and writes of arrays of any length less than 65k bytes, provided that they start on a logical block boundary and that the length is a multiple of four bytes. When only part of a block is written, the contents of the remainder of that logical block will be undefined.

C.1.2 Volume Sets

ODS-1 does not support volume sets. A volume set is a collection of related units that are normally treated as one logical device in the usual operating system concept. Each unit contains its own Files-11 structure. However, files on the various units in a volume set may be referenced with a relative volume number, which uniquely determines which unit in the set the file is located on. Other sections in this specification will make occasional reference to volume sets and relative volume numbers where hooks for their implementation exist. Since volume sets have not been implemented as yet, however, no complete specification is provided here.

C.2 FILES

Any data in a volume or volume set that is of any interest (i.e., all blocks not available for allocation) is contained in a file. A file is an ordered set of virtual blocks, where a virtual block is an array of 512 8 bit bytes. The virtual blocks of a file are consecutively numbered from 1 to n, where n blocks have been allocated to the file. The number assigned to a virtual block is called (obviously) its virtual block number, or VBN. Each virtual block is mapped to a unique logical block in the volume set by Files-11. Virtual blocks may be processed in the same manner as logical blocks. Any array of bytes less than 65k in length may be read or written, provided that the transfer starts on a virtual block boundary and that its length is a multiple of four.

C.2.1 File ID

Each file in a volume set is uniquely identified by a File ID. A File ID is a binary value consisting of 48 bits (3 PDP-11 words). It is supplied by the file system when the file is created, and must be supplied by the user whenever he wishes to reference a particular file.

The three words of the File ID are used as follows:

- Word 1 - File Number

  Locates the file within a particular unit of the volume set. File numbers must lie in the range 1 through 65535. The set of file numbers on a unit is moderately (but not totally) dense. At any instant in time, a file number uniquely identifies one file within that unit.

- Word 2 - File Sequence Number

  Identifies the current use of an individual file number on a unit. File numbers are re-used. When a file is deleted its file number becomes available for future use for some other file. Each time a file number is re-used, a different file sequence number is assigned to distinguish the uses of that file number. The file sequence number is essential since it is perfectly legal for users to remember and attempt to use a File ID long after that file has been deleted.

- Word 3 - Relative Volume Number

  Identifies which unit of a volume set the file is located on. Volume sets are at present not implemented. The only legal value for the relative volume number in any context is zero.

C.2.2 File Header

Each file on a Files-11 volume is described by a file header. The file header is a block that contains all the information necessary to access the file. It is not part of the file. It is contained in the volume's index file. (The index file is described in Section C.4.1. The header block is organized into four areas, of which the first three are variable in size.

C.2.2.1 Header Area - The information in the header area permits the file system to verify that this block is in fact a file header and, in particular, is the header being sought by the user. It contains the file number and file sequence number of the file, as well as its ownership and protection codes. This area also contains offsets to the other areas of the file header, thus defining their size. Finally, the header area contains a user attribute area, which may be used by the user to store a limited amount of data describing the file.

C.2.2.2 Ident Area - The ident area of a file header contains identification and accounting data about the file. Stored here are the primary name of the file, its creation date and time, revision count, date, and time, and expiration date.

C.2.2.3 Map Area - The map area describes the mapping of virtual blocks of the file to the logical blocks of the volume. The mapping data consists of a list of retrieval pointers. Each retrieval pointer describes one logically contiguous segment of the file. The map area also contains the linkage to the next extension header of the file, if such exists.

C.2.2.4 End Checksum - The last two bytes of the file header contain a 16 bit additive checksum of the remaining 255 words of the file header. The checksum is used to help verify that the block is in fact a file header.

C.2.3 Extension Headers

Since the file header is of fixed size, it is inevitable that for some files the mapping information will not fit in the allocated space. A file with a large amount of mapping data is therefore represented with a chain of file headers. Each header maps a consecutive set of virtual blocks. The extension linkage in the map area links the headers together in order of ascending virtual block numbers.

Multiple headers are also needed for files that span units in a volume set. A header may only map logical blocks located on its unit. Therefore, a multi-volume file is represented by headers on all units that contain portions of that file.

C.2.4 File Header - Detailed Description

This section describes in detail the items contained in the file header. Each item is identified by a symbol which represents the offset address of that item within its area in the file header. Any item may be located in the file header by locating the area to which it belongs and then adding the value of its offset address. Users who concern themselves with the contents of file headers are strongly urged to use the offset symbols. The symbols may be defined in assembly language programs by calling and invoking the macro FHDOF$, which may be found in the macro library of any system that supports Files-11.

C.2.4.1 Header Area Description - The header area of the file header always starts at byte 0. It contains the basic information needed for checking the validity of accesses to the file.

H.IDOF: 1 Byte - Ident Area Offset

This byte contains the number of 16 bit words between the start of the file header and the start of the ident area. It defines the location of the ident area and the size of the header area.

H.MPOF: 1 Byte - Map Area Offset

This byte contains the number of 16 bit words between the start of the file header and the start of the map area. It defines the location of the map area and, together with H.IDOF, the size of the ident area.

H.FNUM: 2 Bytes - File Number

This word contains the file number of the file.

H.FSEQ: 2 Bytes - File Sequence Number

This word contains the file sequence number of the file.

H.FLEV: 2 Bytes - File Structure Level

The file structure level is used to identify different versions of Files-11 as they affect the structure of the file header. This permits upwards compatibility of file structures as Files-11 evolves, in that the structure level word identifies the version of Files-11 that created this particular file. This document describes version 1 of Files-11. The only legal contents for H.FLEV is 401 octal.

H.FOWN: 2 Bytes - File Owner UIC

H.PROG = H.FOWN+0 Programmer (Member) Number

H.PROJ = H.FOWN+1 Project (Group) Number

This word contains the binary user identification code (UIC) of the owner of the file. The file owner is usually (but not necessarily) the creator of the file.

H.FPRO: 2 Bytes - File Protection Code

This word controls what access all users in the system may have to the file. Accessors of a file are categorized according to the relationship between the UIC of the accessor and the UIC of the owner of the file. Each category is controlled by a four bit field in the protection word. The category of the accessor is selected as follows:

- System - Bits 0 - 3

  The accessor is subject to system protection if the project number of the UIC under which he is running is 10 octal or less.

- Owner - Bits 4 - 7

  The accessor is subject to owner protection if the UIC under which he is running exactly matches the file owner UIC.

- Group - Bits 8 - 11

  The accessor is subject to group protection if the project number of his UIC matches the project number of the file owner UIC.

- World - Bits 12 - 15

  The accessor is subject to world protection if he does not fit into any of the above categories.

  Four types of access intents are defined in Files-11: read, write, extend, and delete. Each four bit field in the protection word is bit encoded to permit or deny any combination of the four types of access to that category of accessors.

Setting a bit denies that type of access to that category. The bits are defined as follows (these values apply to a right-justified protection field):

- FP.RDV Deny read access
- FP.WRV Deny write access
- FP.EXT Deny extend access
- FP.DEL Deny delete access

When a user attempts to access a file, protection checks are performed in all the categories to which he is eligible, in the order system - owner - group - world. The user is granted access to the file if any of the categories to which he is eligible grants him access.

H.FCHA: 2 Bytes - File Characteristics

H.UCHA = H.FCHA+0 User Controlled Char.
H.SCHA = H.FCHA+1 System Controlled Char.

The user controlled characteristics byte contains the following flag bits:

-	1 Bit, Reserved.
UC.NID	Set if incremental dump (backup) is to be disabled for this file.
UC.WBC	Set if the file is to be write-back cached. For example, if a cache is used for the file data, data written by a user is only written back to the disk when is it removed from the cache. Clear for write-through cache operation.
UC.RCK	Set if the file is to be read-checked. All read operations on the file, including reads of the file header(s), will be performed with a read, read-compare to assure data integrity.
UC.WCK	Set if the file is to be write-checked. All write operations on the file, including modifications of the file

header(s), will be performed with a write, read-compare to assure data integrity.

UC.CNB	Set if the file is allocated contiguous best effort. In other words, as contiguous as possible.
UC.DLK	Set if the file is deaccess-locked. This bit is used as a flag warning that the file was not properly closed and may contain inconsistent data. Access to the file is denied if this bit is set.
UC.CON	Set if the file is logically contiguous. For example, if for all virtual blocks in the file, virtual block i maps to logical block k+i on one unit for some constant k. This bit may be implicitly set or cleared by file system operations that allocate space to the file. The user may only clear it explicitly.

The system controlled characteristics byte contains the following flag bits:

-	3 Bits, Reserved.
-	Reserved (Access Control List).
SC.SPL	Set if the file is an intermediate file for spooling.
SC.DIR	Set if the file is a directory.
SC.BAD	Set if there is a bad data block in the file. This bit is as yet unimplemented. It is intended for dynamic bad block handling.
SC.MDL	Set if the file is marked for delete. If this bit is set, further accesses to the file are denied, and the file will be physically deleted when no users are accessing it.

H.UFAT: 32 Bytes - User Attribute Area

This area is intended for the storage of a limited quantity of "user file attributes", i.e., any data the user deems useful for processing the file that is not part of the file itself. An example of the use of the user attribute area is presented in Section C.5.1 (FCS File Attributes).

S.HDHD: 46 Bytes - Size of Header Area

This symbol represents the total size of the header area containing all of the above entries.

C.2.4.2 Ident Area Description - The ident area of the file header begins at the word indicated by H.IDOF. It contains identification and accounting data about the file.

I.FNAM: 6 Bytes - File Name

These three words contain the name of the file, packed three Radix-50 characters to the word. This name usually, but not necessarily, corresponds to the name of the file's primary directory entry.

I.FTYP: 2 Bytes - File Type

This word contains the type of the file in the form of three Radix-50 characters.

I.FVER: 2 Bytes - Version Number

This word contains the version number of the file in binary form.

I.RVNO: 2 Bytes - Revision Number

This word contains the revision count of the file. The revision count is the number of times the file has been accessed for write.

I.RVDT: 7 Bytes - Revision Date

The revision date is the date on which the file was last deaccessed after being accessed for write. It is stored in ASCII in the form "DDMMMYY", where DD is two digits representing the day of the month, MMM is three characters representing the month, and YY is the last two digits of the year.

I.RVTI: 6 Bytes - Revision Time

The revision time is the time of day on which the file was last deaccessed after being accessed for write. It is stored in ASCII in the format "HHMMSS", where HH is the hour, MM is the minute, and SS is the second.

I.CRDT: 7 Bytes - Creation Date

These seven bytes contain the date on which the file was created. The format is the same as that of the revision date above.

I.CRTI: 6 Bytes - Creation Time

These six bytes contain the time of day at which the file was created. The format is the same as that of the revision time above.

I.EXDT: 7 Bytes - Expiration Date

These seven bytes contain the date on which the file becomes eligible to be deleted. The format is the same as that of the revision and creation dates above.

- : 1 Byte - (unused)

This unused byte is present to round up the size if the ident area to a word boundary.

S.IDHD: 46 Bytes - Size of Ident Area

This symbol represents the size of the ident area containing all of the above entries.

C.2.4.3 Map Area Description - The map area of the file header starts at the word indicated by H.MPOF. It contains the information necessary to map the virtual blocks of the file to the logical blocks of the volume.

M.ESQN: 1 Byte - Extension Segment Number

This byte contains the value n, where this header is the n+1th header of the file. In other words, headers of a file are numbered sequentially starting with 0.

M.ERVN: 1 Byte - Extension Relative Volume No.

This byte contains the relative volume number of the unit in the volume set that contains the next sequential extension header for this file. If there is no extension header, or if the extension header is located on the same unit as this header, this byte contains 0.

M.EFNU: 2 Bytes - Extension File Number

This word contains the file number of the next sequential extension header for this file. If there is no extension header, this word contains 0.

M.EFSQ: 2 Bytes - Extension File Sequence Number

This word contains the file sequence number of the next sequential extension header for this file. If there is no extension header, this word contains 0.

M.CTSZ: 1 Byte - Block Count Field Size

This byte contains a count of the number of bytes used to represent the count field in the retrieval pointers in the map area. The retrieval pointer format is described under M.RTRV below.

M.LBSZ: 1 Byte - LBN Field Size

This byte contains a count of the number of bytes used to represent the logical block number field in the retrieval pointers in the map area. The contents of M.CTSZ and M.LBSZ must add up to an even number.

M.USE: 1 Byte - Map Words In Use

This byte contains a count of the number of words in the map area that are presently occupied by retrieval pointers.

M.MAX: 1 Byte - Map Words Available

This byte contains the total number of words available for retrieval pointers in the map area.

M.RTRV: variable - Retrieval Pointers

This area contains the retrieval pointers that actually map the virtual blocks of the file to the logical blocks of the volume. Each retrieval pointer describes a consecutively numbered group of logical blocks which is part of the file. The count field contains the binary value n to represent a group of n+1 logical blocks. The logical block number field contains the logical block number of the first logical block in the group. Thus each retrieval pointer maps virtual blocks j through j+n into logical blocks k through k+n, respectively, where j is the total number plus one of virtual blocks represented by all preceding retrieval pointers in this and all preceding headers of the file, n is the value contained in the count field, and k is the value contained in the logical block number field.

Although the data in the map area provides for arbitrarily extensible retrieval pointer formats, Files-11 has defined only three. Of these, only the first is currently implemented. The other two are presented out of historical interest. They will never be supported.

Format 1: M.CTSZ = 1
M.LBSZ = 3

The total retrieval pointer length is four bytes. Byte 1 contains the high order bits of the 24 bit LBN. Byte 2 contains the count field, and bytes 3 and 4 contain the low 16 bits of the LBN.

```
|----------|----------|
|  Count   |   High   |
|----------|-        -|
|    Low Order LBN    |
|---------------------|
```

Format 2: M.CTSZ = 2
M.LBSZ = 2

The total retrieval pointer length is four bytes. The first word contains a 16 bit count field and the second word contains a 16 bit LBN field.

```
|---------------------|
|       Count         |
|---------------------|
|         LBN         |
|---------------------|
```

Format 3: M.CTSZ = 2
M.LBSZ = 4

The total retrieval pointer length is six bytes. The first word contains a 16 bit count field and the second and third words contain a 32 bit LBN field.

```
|---------------------|
|    Count            |
|---------------------|
|      High           |
|--          LBN    --|
|      Low            |
|---------------------|
```

S.MPHD: 10 Bytes - Size of Map Area

This symbol represents the size of the map area, not including the space used for the retrieval pointers.

C.2.4.4 End Checksum Description - The header check sum occupies the last two bytes of the file header. It is verified every time a header is read, and is recomputed every time a header is written.

H.CKSM: 2 Bytes - Block Checksum

This word is a simple additive checksum of all other words in the block. It is computed by the following PDP-11 routine or its equivalent:

```
        MOV     Header-address,R0
        CLR     R1
        MOV     #255.,R2
10$:     ADD     (R0)+,R1
        SOB     R2,10$
        MOV     R1,(R0)
```

C.2.4.5 File Header Layout -
Header Area

```
          |--------------------|--------------------|
 H.MPOF   |  Map Area Offset   | Ident Area Offset  |   H.IDOF
          |--------------------|--------------------|
          |               File Number               |   H.FNUM
          |-----------------------------------------|
          |          File Sequence Number           |   H.FSEQ
          |-----------------------------------------|
          |          File Structure Level           |   H.FLEV
          |-----------------------------------------|   H.FOWN
 H.PROJ   |             File Owner UIC              |   H.PROG
          |-----------------------------------------|
          |             File Protection             |   H.FPRO
          |--------------------|--------------------|   H.FCHA
 H.SCHA   |    System Char.    |     User Char.     |   H.UCHA
          |--------------------|--------------------|
          |                                         |   H.UFAT
          |                                         |
          |                                         |
          |           User Attribute Area           |
          |                                         |
          |                                         |
          |                                         |
          |-----------------------------------------|   S.HDHD
```

FILES

Ident Area

```
          |----------------------------------------|
          |                                        |   I.FNAM
          |--                                    --|
          |               File Name                |
          |--                                    --|
          |                                        |
          |----------------------------------------|
          |               File Type                |   I.FTYP
          |----------------------------------------|
          |             Version Number             |   I.FVER
          |----------------------------------------|
          |            Revision Number             |   I.RVNO
          |----------------------------------------|
          |                                        |   I.RVDT
          |--                                    --|
          |             Revision Date              |
          |--                                    --|
          |                                        |
          |-------------------|                  --|
I.RVTI    |                   |                    |
          |--                 |--------------------|
          |                                        |
          |--           Revision Time            --|
          |                                        |
          |-------------------|                  --|
I.CRDT    |                   |                    |
          |--                 |--------------------|
          |                                        |
          |--                                    --|
          |             Creation Date              |
          |--                                    --|
          |                                        |
          |----------------------------------------|
          |                                        |   I.CRTI
          |--                                    --|
          |             Creation Time              |
          |--                                    --|
          |                                        |
          |----------------------------------------|
          |                                        |   I.EXDT
          |--                                    --|
          |            Expiration Date             |
          |--                                    --|
          |                                        |
          |-------------------|                  --|
          |    (not used)     |                    |
          |-------------------|--------------------|   S.IDHD
```

Map Area

```
            |-------------------|-------------------|
  M.ERVN    |   Extension RVN   |  Ext. Seg. Num.   |    M.ESQN
            |-------------------|-------------------|
            |        Extension File Number          |    M.EFNU
            |---------------------------------------|
            |      Extension File Seq. Num.         |    M.EFSQ
            |-------------------|-------------------|
  M.LBSZ    |  LBN Field Size   | Count Field Size  |    M.CTSZ
            |-------------------|-------------------|
  M.MAX     | Map Words Avail.  | Map Words in Use  |    M.USE
            |-------------------|-------------------|    S.MPHD
            |                                       |    M.RTRV
            |                                       |
            |                                       |
            |         Retrieval Pointers            |
            |                                       |
            |                                       |
            |                                       |
            |---------------------------------------|
            |        File Header Checksum           |    H.CKSM
            |---------------------------------------|
```

C.3 DIRECTORIES

Files-11 provides directories to allow the organization of files in a meaningful way. While the File ID is sufficient to locate a file uniquely on a volume set, it is hardly mnemonic. Directories are files whose sole function is to associate file name strings with File ID's.

C.3.1 Directory Hierarchies

Since directories are files with no special attributes, directories may list files that are in turn directories. Thus, an operating system may construct directory hierarchies of arbitrary depth and complexity.

C.3.1.1 User File Directories - This implementation of Files-11 supports a multilevel directory hierarchy. This hierarchy is a simple tree structure with *root* as the master file directory (MFD). There are two forms of valid directory names: alphanumeric and numeric. Alphanumeric directory names can be any legal file name consisting of up to nine characters. Alphanumeric directory names are created in the form "yyyyyyyyy.DIR;1." Numeric directory names consist of two three-digit octal numbers, in the range 0 through 377. If fewer than nine characters are used, the name is padded with leading zeros. Numeric directory names are created in the form "nnnmmm.DIR;1."

C.3.2 Directory Structure

A directory is a file consisting of 16 byte records. It is structured as an FCS fixed length record file, with no carriage control attributes (see Section C.5 for a description of FCS files). Each record is a directory entry. The entries are not required to be ordered, or densely packed, nor do they have any other relationship to each other, except that no two entries in one directory may contain the same name, type, and version. Each entry contains the following:

File ID	The three word binary File ID of the file that this directory entry represents. If the file number portion of the File ID field is zero, then this record is empty and may be used for a new directory entry.
Name	The name of the file may be up to 9 characters. It is stored as three words, each containing three Radix-50 packed characters.
Type	The type of the file (also historically referred to as the extension) may be up to three characters. It is stored as one word of Radix-50 packed characters.
Version	The version number of the file is stored in binary in one word.

```
|---------------------|
|                     |
|--                 --|
|       File ID       |
|--                 --|
|                     |
|---------------------|
|                     |
|--                 --|
|        Name         |
|--                 --|
```

```
|                      |
|----------------------|
|         Type         |
|----------------------|
|        Version       |
|----------------------|
```

C.3.3 Directory Protection

Since directories are files with no special characteristics, they may be accessed like all other files, and are subject to the same protection mechanism. However, implementations of Files-11 support three special functions for the management of directories, namely FIND, REMOVE, and ENTER. A user performing such a directory operation must have the following privileges to be allowed the various functions:

Find: READ
Remove: READ, WRITE
Enter: READ, WRITE

Note that the same privilege is required for both enter and remove. The recovery for an operation that involves a remove at the beginning of the sequence is an enter.

C.4 KNOWN FILES

Clearly any file system must maintain some data structure on the medium which is used to control the file organization. In Files-11 this data is kept in five files. These files are created when a new volume is initialized. They are unique in that their File ID's are known constants. These five files have the following uses:

File ID 1,1,0 is the index file. The index file is the root of the entire Files-11 structure. It contains the volume's bootstrap block and the home block, which is used to identify the volume and locate the rest of the file structure. The index file also contains all of the file headers for the volume, and a bitmap to control the allocation of file headers.

File ID 2,2,0 is the storage bitmap file. It is used to control the allocation of logical blocks on the volume.

File ID 3,3,0 is the bad block file. It is a file containing all of the known bad blocks on the volume.

File ID 4,4,0 is the volume master file directory (or MFD). It forms the root of the volume's directory structure. The MFD lists the five known files, all first level user directories, and whatever other files the user chooses to enter.

File ID 5,5,0 is the system core image file. Its use is operating system dependent. Its basic purpose is to provide a file of known File ID for the use of the operating system.

C.4.1 Index File

The index file is File ID 1,1,0. It is listed in the MFD as INDEXF.SYS;1. The index file is the root of the Files-11 structure in that it provides the means for identification and initial access to a Files-11 volume, and contains the access data for all files on the volume (including itself).

C.4.1.1 Bootstrap Block - Virtual block 1 of the index file is the volume's boot block. It is always mapped to logical block 0 of the volume. If the volume is the system device of an operating system, the boot block contains an operating system dependent program which reads the operating system into memory when the boot block is read and executed by a machine's hardware bootstrap. If the volume is not a system device, the boot block contains a small program that outputs a message on the system console to inform the operator to that effect.

C.4.1.2 Home Block - Virtual block 2 of the index file is the volume's home block. The logical block containing the home block is the first good block on the volume out of the sequence 1, 256, 512, 768, 1024, 1280, ... 256n. The purpose of the home block is to identify the volume as Files-11, establish the specific identity of the volume, and serve as the ground zero entry point into the volume's file structure. The home block is recognized as a home block by the presence of checksums in known places and by the presence of predictable values in certain locations.

Items contained in the home block are identified by symbolic offsets in the same manner as items in the file header. The symbols may be defined in assembly language programs by calling and invoking the macro HMBOF$, which may be found in the macro library of any system that supports Files-11.

H.IBSZ: 2 Bytes - Index File Bitmap Size

This 16 bit word contains the number of blocks that make up the index file bitmap. (The index file bitmap is discussed in Section C.4.1.3. This value must be non-zero for a valid home block.

H.IBLB: 4 Bytes - Index File Bitmap LBN

This double word contains the starting logical block address of the index file bitmap. Once the home block of a volume has been found, it is this value that provides access to the rest of the index file and to the volume. The LBN is stored with the high order in the first 16 bits, followed by the low order portion. This value must be non-zero for a valid home block.

H.FMAX: 2 Bytes - Maximum Number of Files

This word contains the maximum number of files that may be present on the volume at any time. This value must be non-zero for a valid home block.

H.SBCL: 2 Bytes - Storage Bitmap Cluster Factor

This word contains the cluster factor used in the storage bitmap file. The cluster factor is the number of blocks represented by each bit in the storage bitmap. **Volume clustering can not be implemented in**

ODS-1. The only legal value for this item is 1.

H.DVTY: 2 Bytes - Disk Device Type

This word is an index identifying the type of disk that contains this volume. It is currently not used and always contains 0.

H.VLEV: 2 Bytes - Volume Structure Level

This word identifies the volume's structure level. Like the file structure level, this word identifies the version of Files-11 which created this volume and permits upwards compatibility of media as Files-11 evolves. The volume structure level is affected by all portions of the Files-11 structure except the contents of the file header. This document describes Files-11 version 1. The only legal values for the structure level are 401 and 402 octal. The former (401) is the standard value for most volumes. The latter (402) is an advisory that the volume contains a multiheader index file. (A multiheader index file is required to support more than about 26,000 files. The index file may in fact be multiheader without the volume having a structure level of 402).

H.VNAM: 12 Bytes - Volume Name

This area contains the volume label as an ASCII string. It is padded out to 12 bytes with nulls. The volume label is used to identify individual volumes.

- : 4 Bytes - Not Used

H.VOWN: 2 Bytes - Volume Owner UIC

This word contains the binary UIC of the owner of the volume. The format is the same as that of the file owner UIC stored in the file header.

H.VPRO: 2 Bytes - Volume Protection Code

This word contains the protection code for the entire volume. Its contents are coded in the same manner as the file protection code stored in the file header, and it is interpreted in the same way in conjunction with the volume owner UIC. All operations on all files on the volume must pass both the volume and the file protection check to be permitted. (Refer to the discussion on file protection described earlier under H.FPRO.

H.VCHA: 2 Bytes - Volume Characteristics

This word contains bits which provide additional control over access to the volume. The following bits are defined:

- CH.NDC - Obsolete, used by RSX-11D/IAS. Set if device control functions are not permitted on this volume. Device control functions are those which can threaten the integrity of the volume, such as direct reading and writing of logical blocks, etc.
- CH.NAT - Obsolete, used by RSX-11D/IAS. Set if the volume may not be attached, i.e., reserved for the sole use by one task.
- CH.SDI - Set if the volume contains only a single directory. If this bit is set, no directories should be created on the volume other than the MFD. The access methods should also be informed of this situation, e.g. by setting the DV.SDI bit in the device characteristics word.

H.DFPR: 2 Bytes - Default File Protection

This word contains the file protection that will be assigned to all files created on this volume if no file protection is specified by the user.

- : 6 Bytes - Not Used

H.WISZ: 1 Byte - Default Window Size

This byte contains the number of retrieval pointers that will be used for the "window" (in core file access data) when files are accessed on the volume, if not otherwise specified by the accessor.

H.FIEX: 1 Byte - Default File Extend

This byte contains the number of blocks that will be allocated to a file when a user extends the file and asks for the system default value for allocation.

H.LRUC: 1 Byte - Directory Pre-access Limit

This byte contains a count of the number of directories to be stored in the file system's directory access cache. More generally, it is an estimate of the number of concurrent users of the volume and its use may be generalized in the future.

H.REVD: 7 Bytes - Date of Last Home Block Revision

This field is in the standard ASCII date format and reflects the date of the last modifications to fields in the home block.

H.REVC: 2 Bytes - Count of Home Block Revisions

This field reflects the number of above mentioned modifications.

- : 2 Bytes - Not Used

H.CHK1: 2 Bytes - First Checksum

This word is an additive checksum of all entries preceding in the home block (i.e., all those listed above). It is computed by the same sort of algorithm as the file header checksum (see H.CKSM).

H.VDAT: 14 Bytes - Volume Creation Date

This area contains the date and time that the volume was initialized. It is in the format "DDMMMYYHHMMSS", followed followed by a single null. (The same format is used in the ident area of the file header, Section C.2.4.2.

- : 382 Bytes Not Used

This area is reserved for the relative volume table for volume sets. This field will not be used, although some versions of DSC referenced this area.

H.PKSR: 4 Bytes - Pack Serial Number

This area contains the manufacturer supplied serial number for the physical volume. For last track devices, the pack serial number is contained on the volume in the manufacturer data. For other devices the user must supply this information manually. The serial number is contained in the home block for convenience and consistency.

- : 12 Bytes - Not Used

This field is reserved for the volume set name.

•
•

H.INDN: 12 Bytes - Volume Name

This area contains another copy of the ASCII volume label. It is padded out to 12 bytes with spaces.

•
•

H.INDO: 12 Bytes - Volume Owner

This area contains an ASCII expansion of the volume owner UIC in the form "[proj,prog]". Both numbers are expressed in decimal and are padded to three digits with leading zeroes. The area is padded out to 12 bytes with trailing spaces.

•
•

H.INDF: 12 Bytes - Format Type

This field contains the ASCII string "DECFILE11A" padded out to 12 bytes with spaces. It identifies the volume as being of Files-11 format.

•
•

- : 2 Bytes - Not Used

H.CHK2: 2 Bytes - Second Checksum

This word is the last word of the home block. It contains an additive checksum of the preceding 255 words of the home block, computed according to the algorithm listed under H.CKSM.

Home Block Layout

```
|---------------------------------------|
|        Index File Bitmap Size         |     H.IBSZ
|---------------------------------------|
|              Index File               |     H.IBLB
|--                                   --|
|              Bitmap LBN               |
|---------------------------------------|
|        Maximum Number of Files        |     H.FMAX
|---------------------------------------|
|     Storage Bitmap Cluster Factor     |     H.SBCL
|---------------------------------------|
```

```
          |          Disk Device Type              |   H.DVTY
          |----------------------------------------|
          |       Volume Structure Level           |   H.VLEV
          |----------------------------------------|
          |                                        |   H.VNAM
          |--                                    --|
          |                                        |
          |--                                    --|
          |             Volume Name                |
          |--                                    --|
          |                                        |
          |--                                    --|
          |                                        |
          |--                                    --|
          |                                        |
          |----------------------------------------|
          |                                        |
          |--           (not used)               --|
          |                                        |
          |----------------------------------------|
          |          Volume Owner UIC              |   H.VOWN
          |----------------------------------------|
          |          Volume Protection             |   H.VPRO
          |----------------------------------------|
          |       Volume Characteristics           |   H.VCHA
          |----------------------------------------|
          |       Default File Protection          |   H.DFPR
          |----------------------------------------|
          |                                        |
          |--                                    --|
          |             (not used)                 |
          |--                                    --|
          |                                        |
          |-------------------|--------------------|
  H.FIEX  | Def. File Extend  | Def. Window Size   |   H.WISZ
          |-------------------|--------------------|
  H.REVD  |                   |  Directory Limit   |   H.LRUC
          |--                 |--------------------|
          |                                        |
          |--                                    --|
          |      Volume Modification Date          |
          |--                                    --|
          |                                        |
          |----------------------------------------|
          |      Volume Modification Count         |   H.REVC
          |----------------------------------------|
          |             (not used)                 |
          |----------------------------------------|
          |            First Checksum              |   H.CHK1
          |----------------------------------------|
          |                                        |   H.VDAT
```

```
|--                                   --|
|                                       |
|--                                   --|
|                                       |
|--                                   --|
|         Volume Creation Date          |
|--                                   --|
|                                       |
|--                                   --|
|                                       |
|--                                   --|
|                                       |
|---------------------------------------|
|                                       |
|                                       |
|                                       |
|                                       |
|              (not used)               |
|                                       |
|                                       |
|                                       |
|                                       |
|---------------------------------------|
|                                       |   H.PKSR
|--       Pack Serial Number            --|
|                                       |
|---------------------------------------|
|---------------------------------------|
|                                       |
|--                                   --|
|                                       |
|--                                   --|
|              (not used)               |
|--                                   --|
|                                       |
|--                                   --|
|                                       |
|--                                   --|
|                                       |
|---------------------------------------|
|                                       |   H.INDN
|--                                   --|
|                                       |
|--                                   --|
|              Volume Name              |
|--                                   --|
|                                       |
|--                                   --|
|                                       |
|--                                   --|
|                                       |
```

```
|----------------------------------------|
|                                        |   H.INDO
|--                                    --|
|                                        |
|--                                    --|
|              Volume Owner              |
|--                                    --|
|                                        |
|--                                    --|
|                                        |
|--                                    --|
|                                        |
|----------------------------------------|
|                                        |   H.INDF
|--                                    --|
|                                        |
|--                                    --|
|              Format Type               |
|--                                    --|
|                                        |
|--                                    --|
|                                        |
|--                                    --|
|                                        |
|----------------------------------------|
|               (not used)               |
|----------------------------------------|
|            Second Checksum             |   H.CHK2
|----------------------------------------|
```

C.4.1.3 Index File Bitmap - The index file bitmap is used to control the allocation of file numbers (and hence file headers). It is simply a bit string of length n, where n is the maximum number of files permitted on the volume (contained in offset H.FMAX in the home block). The bitmap spans over as many blocks as is necessary to hold it, i.e., maximum number of files divided by 4096 and rounded up. The number of blocks in the bitmap is contained in offset H.IBSZ of the home block.

The bits in the index file bitmap are numbered sequentially from 0 to n-1 in the obvious manner, i.e., from right to left in each byte, and in order of increasing byte address. Bit j is used to represent file number j+1: if the bit is 1, then that file number is in use. If the bit is 0, then that file number is not in use and may be assigned to a newly created file.

The index file bitmap starts at virtual block 3 of the index file and continues through VBN 2+m, where m is the number of blocks in the bitmap. It is located at the logical block indicated by offset H.IBLB in the home block.

C.4.1.4 File Headers - The rest of the index file contains all the file headers for the volume. The first 16 file headers (for file numbers 1 to 16) are logically contiguous with the index file bitmap to facilitate their location. The rest may be allocated wherever the file system sees fit. Thus the first 16 file headers may be located from data in the home block (H.IBSZ and H.IBLB) while the rest must be located through the mapping data in the index file header. The file header for file number n is located at virtual block 2+m+n (where m is the number of blocks in the index file bitmap).

C.4.2 Storage Bitmap File

The storage bitmap file is File ID 2,2,0. It is listed in the MFD as BITMAP.SYS;1. The storage bitmap is used to control the available space on a unit. It consists of a storage control block which contains summary information about the unit, and the bitmap itself which lists the availablilty of individual blocks.

C.4.2.1 Storage Control Block - Virtual block 1 of the storage bitmap is the storage control block. It contains summary information intended to optimize allocation of space on the unit. The storage control block has the following format for disks with less than 4096*126, (516,096) blocks:

(3 bytes)	Not used (zero)
(1 byte)	Number of storage bitmap blocks (less than 127)
(2 bytes)	Number of free blocks in 1st bitmap block
(2 bytes)	Free block pointer in 1st bitmap block
	.
	.
	.
(2 bytes)	Number of free blocks in nth bitmap block
(2 bytes)	Free block pointer in nth bitmap block
(4 bytes)	Size of the unit in logical blocks

For larger disks the following format is used:

(3 bytes)	Not used (zero)
(1 byte)	Number of storage bitmap blocks (greater than 126)
(4 bytes)	Size of the unit in logical blocks
(246 bytes)	Not used (zero)

NOTE

> Current implementations of Files-11 do not correctly initialize the word pairs containing number of free blocks and free block pointer for each bitmap block, nor are these values maintained as space is allocated and freed on the unit. They are therefore best looked upon as 2n garbage words and should not be used by future implementations of Files-11 until the disk structure is formally updated.

C.4.2.2 Storage Bitmap - Virtual blocks 2 through n+1 are the storage bitmap itself. It is best viewed as a bit string of length m, numbered from 0 to m-1, where m is the total number of logical blocks on the unit rounded up to the next multiple of 4096. The bits are addressed in the usual manner (packed right to left in sequentially numbered bytes). Since each virtual block holds 4096 bits, n blocks, where n = m/4096, are used to hold the bitmap. Bit j of the bitmap represents logical block j of the volume. If the bit is set, the block is free. If clear, the block is allocated. Clearly the last k bits of the bitmap are always clear, where k is the difference between the true size of the volume and m, the length of the bitmap.

The size of the bitmap is limited to 256 blocks. In fact, due to existing implementations on all RSX systems, the retrieval pointers must be in one of the following two forms:

1. A single retrieval pointer mapping the entire BITMAP.SYS file.

2. Two retrieval pointers, the first mapping the storage control block only, and the second mapping the entire bitmap proper.

   This restriction limits ODS-1 to a volume of 4096*255 blocks (1,044,480 blocks or about 500 megabytes).

C.4.3 Bad Block File

The bad block file is File ID 3,3,0. It is listed in the MFD as BADBLK.SYS;1. The bad block file is simply a file containing all of the known bad blocks on the volume.

C.4.3.1 Bad Block Descriptor - The last virtual block of the bad block file is the bad block descriptor for the volume. It is always located on the last good block of the volume. This block may contain a listing of the bad blocks on the volume produced by a bad block scan program or diagnostic. The format of the bad block data is identical to the map area of a file header, except that the first four entries (M.ESQN, M.ERVN, M.EFNU, and M.EFSQ) are not present. The last word of the block contains the usual additive checksum. (See earlier in this chapter for a description of the map area.) This block is included in the bad block file to save the data it contains for future re-initialization of the volume.

Bad Block Descriptor Layout

```
|-------------------|-------------------|
|  LBN Field Size   | Count Field Size  |
|-------------------|-------------------|
| Map Words Avail.  | Map Words in Use  |
|-------------------|-------------------|
|                                       |
|                                       |
|                                       |
|          Retrieval Pointers           |
|                                       |
|                                       |
|                                       |
|---------------------------------------|
|            Block Checksum             |
|---------------------------------------|
```

C.4.4 Master File Directory

The master file directory is File ID 4,4,0. It is listed in the MFD (itself) as 000000.DIR;1. The MFD is the root of the volume's directory structure. It lists the five known files, plus whatever the user chooses to enter.

C.4.5 Core Image File

The core image file is File ID 5,5,0. It is listed in the MFD as CORIMG.SYS;1. Its use is operating system dependent. In general, it provides a file of known File ID for the use of the operating system, for use as a swap area, for example, or as a monitor overlay area, etc.

C.5 FCS FILE STRUCTURE

File Control Services (FCS) is a user level interface to Files-11. Its principal feature is a record control facility that allows sequential processing of variable length records and sequential and random access to fixed length record files. FCS interfaces to the virtual block facility provided by the basic Files-11 structure.

C.5.1 FCS File Attributes

FCS stores attribute information about the file in the file's user attribute area. (Refer to the discussion of the H.UFAT byte in Section C.2.4.1.) It uses only the first 7 words. The rest are ignored by FCS, but are reserved by DEC. (RMS uses an additional 3 words, 10 words in all, for relative and indexed file attributes.) The following items are contained in the attribute area. They are identified by the usual symbolic offsets (relative to the start of the attribute area). The offsets may be defined in assembly language programs by calling and invoking the macro FDOFF$ DEF$L. Flag values and bits may be defined by calling and invoking the macro FCSBT$. These macros are in the system macro library of any operating system that supports Files-11. Alternatively, all these values are defined in the system object library of any system that supports Files-11, and may be obtained at link time.

C.5.1.1 F.RTYP: 1 Byte - Record Type - This byte identifies which type of records are contained in this file. The following three values are legal:

R.FIX	Fixed length records.
R.VAR	Variable length records.
R.SEQ	Sequenced Variable Length records

C.5.1.2 F.RATT: 1 Byte - Record Attributes - This byte contains record attribute bits that control the handling of records under various contexts. The following flag bits are defined:

FD.FTN	Use Fortran carriage control if set. The first byte of each record is to be interpreted as a standard Fortran carriage control character when the record is copied to a carriage control device.

FD.CR	Use implied carriage control if set. When the file is copied to a carriage control device, each record is to be preceded by a line feed and followed by a carriage return. Note that the FD.FTN and FD.CR bits are mutually exclusive.
FD.PRN	Used to indicate that the two byte sequence number field for R.SEQ record format is to be interpreted as print control information. See later in this appendix for format of print information.
FD.BLK	Records do not cross block boundaries if set. Generally, there will be dead space at the end of each block. How this is handled is explained in the description of record structure in Section C.5.2.1.

C.5.1.3 F.RSIZ: 2 Bytes - Record Size - In a fixed length record file, this word contains the size of the records in bytes. In a variable or sequenced variable length record file, this word contains the size in bytes of the longest record in the file.

C.5.1.4 F.HIBK: 4 Bytes - Highest VBN Allocated - This 32 bit number is a count of the number of virtual blocks allocated to the file. Since this value is maintained by FCS, it is usually correct, but it is not guaranteed since FCS is a user level package.

C.5.1.5 F.EFBK: 4 Bytes - End of File Block - This 32 bit number is the VBN in which the end of file is located. Both F.HIBK and F.EFBK are stored with the high order half in the first two bytes, followed by the low order half.

C.5.1.6 F.FFBY: 2 Bytes - First Free Byte - This word is a count of the number of bytes in use in the virtual block containing the end of file. In other words, it is the offset to the first byte of the file available for appending. Note that an end of file that falls on a block boundary may be represented in either of two ways. If the file contains precisely n blocks, F.EFBK may contain n and F.FFBY will contain 512., or F.EFBK may contain n+1 and F.FFBY will contain 0.

C.5.1.7 S.FATT: 14 Bytes - Size of Attribute Block - This symbol represents the total number of bytes in the FCS file attribute block.

C.5.1.8 FCS File Attributes Layout -

```
          |-------------------|-------------------|
F.RATT    |   Record Attr.    |    Record Type    |    F.RTYP
          |-------------------|-------------------|
          |          Record Size (Bytes)          |    F.RSIZ
          |---------------------------------------|
          |             Highest VBN               |    F.HIBK
          |--                                   --|
          |              Allocated                |
          |---------------------------------------|
          |             End of File               |    F.EFBK
          |--                                   --|
          |                 VBN                   |
          |---------------------------------------|
          |            First Free Byte            |    F.FFBY
          |---------------------------------------|    S.FATT
```

C.5.2 Record Structure

This section describes how records are packed in the virtual blocks of a disk file. In general, FCS treats a disk file as a sequentially numbered array of bytes. Records are numbered consecutively starting with 1.

C.5.2.1 Fixed Length Records - In a file consisting of fixed length records, the records are simply packed end to end with no additional control information. If the record length is odd, each record is padded with a single byte. The content of the pad byte is undefined. For direct access, the address of a record is computed as follows:

Let: n = record number
k = record size (in bytes)
m = byte address of record in file

q = number of records per block
j = VBN containing the start of the record
i = byte offset within VBN j

then h = ((k+1)/2)2 (rounded up record length)
m = (n-1)h
j = m/512+1 (truncated)
i = m mod 512

The previous discussion assumes that records cross block boundaries (that is, FD.BLK is not set). If records do not cross block boundaries, they are limited to 512 bytes, and the following equations apply (the variables are defined as above):

h = ((k+1)/2)2 (rounded up record length)
q = 512/k (truncated)
j = (n-1)/q+1 (truncated)
i = ((n-1) mod q)h

C.5.2.2 Variable Length Records - In a file consisting of variable length records, records may be up to 32767 bytes in length. Each record is preceded by a two byte binary count of the bytes in the record (the count does not include itself). For example, a null record is represented by a single zero word. The byte count is always word aligned. For example,if a record ends on an odd byte boundary, it is padded with a single byte. The content of the pad byte is undefined.

If records do not cross block boundaries (FD.BLK is set), they are limited to a size of 510 bytes. A byte count of -1 is used as a flag to signal that there are no more records in a particular block. The remainder of that block is then dead space and the next record in the file starts at the beginning of the next block.

C.5.2.3 Sequenced Variable Length Records - The format of a sequenced file is identical to a variable length record file except that a two byte sequence number field is located immediately after the byte count field of each record. This field contains a binary value which is usually interpreted as the line number of that record (see F.RATT, FD.PRN and Section C.5.2.3.1. The sequence number is not returned as part of the data when a record is read, but is available separately. Note that the record byte count field counts the sequence number field as well as the data of the record.

Format of Two Byte Print Control Field in R.SEQ Records - If the FD.PRN bit is set in the record attribute then the two byte "sequence number" field is used to contain carriage control data for the record. Byte 0 is print control information to act upon before the record data is output to a unit record device. Byte 1 is print control information to act upon after the record data has been output to a unit record device.

APPENDIX D

FILES-11 QIO INTERFACE TO THE ACPS

This appendix describes the QIO level interface to the file processors (ACPs). These include F11ACP for Files-11 disks and MTAACP for ANSI magnetic tape.

F11ACP supports the following functions:

IO.CRE	Create file
IO.DEL	Delete file
IO.ACR	Access file for read only
IO.ACW	Access file for read/write
IO.ACE	Access file for read/write/extend
IO.DAC	Deaccess file
IO.EXT	Extend file
IO.RAT	Read file attributes
IO.WAT	Write file attributes
IO.FNA	Find file name in directory
IO.RNA	Remove file name from directory
IO.ENA	Enter file name in directory
IO.ULK	Unlock block

D.1 QIO PARAMETER LIST FORMAT

The device-independent part of a file processing QIO parameter list is identical to all other QIO parameter lists. The file processor QIOs require the following six additional words in the parameter lists:

Parameter Word 1	Address of a 3-word block containing the file identifier
Parameter Word 2	Address of the attribute list
Parameter Words 3 & 4	Size and extend control information
Parameter Word 5	Window size information and access control

Parameter word 6	Address of the file name block

NOTE

The P/OS Executive treats File Identifier Blocks, filename blocks, and attribute list entries as read/write data. For this reason, they may not be used in read-only code segments or libraries.

D.1.1 File Identification Block

The File Identification Block is a 3-word block containing the file number and the file sequence number. The format of the File Identification Block is as follows:

```
---------------------------------
|          File Number          |
|-------------------------------|
|     File Sequence Number      |
|-------------------------------|
|           Reserved            |
---------------------------------
```

F11ACP uses the file number as an index to the file header in the index file. Each time a header block is used for a new file, the file sequence number is incremented. This insures that the file header is always unique. The third word is not currently used but is reserved for the future.

D.1.2 The Attribute List

The file attribute list controls F11ACP reads or writes. File attributes are fields in the file header, described later in this appendix.

The attribute list contains a variable number of entries terminated by a byte containing zero (0). The maximum number of entries in the attribute list is six.

An entry in the attribute list has the following format:

```
.BYTE  <Attribute type>, Attribute size
.WORD  Pointer to the attribute buffer
```

D.1.2.1 The Attribute Type - This field identifies the individual attribute to be read or written. The sign of the attribute type code determines whether the transfer is a read or write operation. If the type code is negative, the ACP reads the attribute into the buffer. If the type code is positive, the ACP writes the attribute to the file header. Note that the sign of the type code must agree with the direction implied by the operation. For example, if the type code is positive, the operation must be an IO.WAT or IO.DAC.

The attribute type is one of the following:

- File owner (H.FOWN)
  The file owner UIC is a binary word. The low byte is the owner number and the high byte is the group number.

- File protection (H.FPRO)
  The file protection word is a bit mask with the following format:

  Each of the fields contains four bits, as follows:

Bit 1	Read Access
Bit 2	Write Access
Bit 3	Extend Access
Bit 4	Delete Access

- File characteristics (H.UCHA)
  The following user characteristics are currently contained in the 1-byte H.UCHA field:

  UC.CON = 200   Logically contiguous file
  UC.DLK = 100   File improperly closed

- Record I/O Area (U.UFAT)
  This field contains a copy of the first seven* words of the file descriptor block.

- File name (I.FNAM)
  The file name is stored as nine Radix-50 characters. the fourth word of this block contains the file type and the fifth word contains the version number.

- File type (I.FTYP)
  The file type is stored as three Radix-50 characters.

- Version number (I.FVER)

---

1. RMS uses 32 bytes. The first seven are compatible with FCS for sequential files.

The version number is stored as a binary number.

- Expiration date (I.EXDT)
  Creation date (I.CRDT)
  Revision date (I.RVDT)
  The expiration date is currently unused. When the file is created, the ACP initializes the creation date to the current date and time. It initializes the expiration and revision dates to 0. The ACP sets the revision date to the current date and time each time the file is deaccessed.

- Statistics block
  This block is described later in this appendix.

- Read entire file header
  This buffer is assumed to be 1000 blocks long. You cannot write this attribute.

- Revision number (I.RVNO)
  The ACP sets the revision number to 0, and increments it every time the file is deaccessed.

- Placement Control

D.1.2.2 Attribute Size - This word specifies the number of bytes of the attribute to be transferred. Legal values are from 1 to the maximum size of the particular attribute. Table D-1 shows the maximum size for each attribute type.

Table D-1: Maximum Size for Each File Attribute

Attribute Type Code	Attribute Type	Maximum Attribute Size in Octal Bytes
1	File owner	6
2	Protection	4
3	File characteristics	2
4	Record I/O area	40

5	File name,type,version number	12
6	File type	4
7	Version number	2
10	Expiration date	7
11	Statistics block	12
12	Entire file header	0
13	Block Size (magtape only)	--
15	Revision number and creation/revision/expiration dates	43
16	Placement control	16

D.1.2.3 Attribute Buffer Address - The attribute Buffer Address field contains the address of the buffer in the user's task space to or from which the attribute is to be transferred.

D.1.3 Size and Extend Control

These two parameters specify how many blocks the file processor allocated to a new file or adds to an existing file. These parameters also control the type of block allocation.

The format is as follows:

```
     .BYTE  <High 8 bits of size>, <extend control>
     .WORD  <Low 16 bits of size>
```

The size field specifies the number of blocks to be allocated to a file on IO.CRE and IO.EXT operations, and the final file size on IO.DEL operations.

The extend control field controls the manner in which an extend operation is to be done. The following bits are defined:

EX.AC1=1	The extend size is to be added as a contiguous block.
EX.AC2=2	Extend by the largest available contiguous piece up to the specified size.

EX.FCO=4	The file must end up contiguous.
EX.ADF=10	Use the default rather than the specified size. The default extend size is the size that was specified when the volume was mounted.
EX.ALL=20	Placement control (see Section D.2.
EX.ENA=200	Enable extend.

D.1.4 Window Size and Access Control

This parameter specifies the window size and access control information in the following format:

```
.BYTE  <window size>, <access control>
```

This word is only processed if the high bit of the access control byte (AC.ENB) is set.

Window size is the number of mapping entries. Specifying a negative window size minimizes window turns. If this byte is zero, the file processor uses the volume default. The size of the window allocated in the dynamic storage region is 6 times the number of mapping entries (each mapping entry is 3 words), plus 10 bytes for the window control block. The mapping entries are allocated in secondary pool. The window control block and a pointer to secondary pool are located in primary pool.

The following access control bits are defined:

AC.LCK=1	Lock out further accesses for Write or Extend
AC.DLK=2	Enable Deaccess lock The deaccess lock sets the lock bit in the file header if the file is deaccessed as the result of a task exit without explicitly deaccessing the file. The lock bit is set by the executive. The lock bit is not set when the system crashes.
AC.LKL=4	Enable block locking
AC.EXL=10	Enable explicit block unlocking
AC.WCK=40	Initiate driver write-checking
AC.ENB=200	Enable Access

NOTE

Both AC.LKL and AC.EXL must be set if you want block locking. If you do not want block locking, both bits must be clear. Any other combination is an error.

D.1.5 File Name Block Pointer

This word contains the address of a 15-word block in the issuing task's space. This block is called the file name block. The file name block is described in detail later in this appendix.

The fields of the file name block that are particularly important in file-processing operations are:

- o Directory identification (N.DID)
  This field is required for all disk operations. It specifies the directory to which the operation applies. This field is not used for tape operations.

- o File identification (N.FID)
  This field is required as input for enter operations. This field is returned as output by find and remove operations.

- o File name (N.FNAM), type (N.FTYP), and version number (N.FVER)
  These fields are required as input to enter, find, and remove operations. For find and remove operations, the file processor locates the appropriate entry by matching the information in these fields with the directory entries.

- o Status word (N.STAT)

- o Wildcard context (N.NEXT)
  This field is required as input for wildcard operations. It specifies the point at which to resume processing. It is updated for the next operation. It must initially be set to 0.

D.2 PLACEMENT CONTROL

The placement control attribute list entry controls the placement of a file in a particular place on the disk. You can specify either exact or approximate placement on IO.CRE and IO.EXT operations.

PLACEMENT CONTROL

The placement control entry must be the first entry in the attribute list.

The format of the placement control attribute list entry is as follows:

```
.BYTE  placement control,0
.WORD  high-order bits of VBN or LBN
.WORK  low-order bits of VBN or LBN
.BLKW  4       ; Buffer to receive starting and ending LBN if
                 AL.LBN is set.
```

The following bits are defined for the placement control field:

AL.VBN=1	Set if block specified is a VBN; otherwise, the block is the LBN
AL.APX=2	Set if you want approximate placement; otherwise, placement is exact
AL.LBN=4	Set if you want starting and ending LBN information

D.3 BLOCK LOCKING

Block locking only occurs when the user accesses a file with AC.LKL and AC.EXL set in the access control byte of the parameter list. Any read or write operation causes a check to see if the block is locked.

A write access locks a block for exclusive access. A write operation can only access a block that is not locked by any accessor. The only exception to this is an exact match with a previous lock owned by the same accessor.

A read access locks a block for shared access. A read operation can access any block locked for shared access.

The user must unlock a block with an explicit unlock request, IO.ULK. IO.ULK may be used to unlock one or all blocks.

If all accessors to a file have not requested block locking, the ACP returns an error.

When the file is deaccessed, all locks owned by the accessor are released.

Each active lock requires eight bytes from the dynamic storage region. This storage is deallocated when the file is deaccessed.

D.4 SUMMARY OF F11ACP FUNCTIONS

The following is a summary of the functions implemented in F11ACP. A list of accepted parameters follows each function. All parameters are required unless specified as optional. Parameters other than those listed are illegal for that function and must be 0.

IO.CRE	Create file	
	#1	The file identifier block is filled in with the file identifier and sequence number of the created file.
	#2	Write Attribute and/or Placement Control list (optional)
	#3 & #4	Extend Control (optional) The amount allocated to the file is returned in the high byte of IOST(1) plus IOST(2).
	#5	May be nonzero but must be disabled
IO.DEL		Delete or truncate file
	#1	Optional if the file is accessed
	#3 & #4	Size to truncate the file to. If not enabled, the file is deleted. If enabled, the remaining 31 bits specify the size the file is to be after truncation. The change in file allocation is returned in the high byte of IOST(1) plus IOST(2). This amount will be zero or negative.
IO.ACR		Access file for read only
IO.ACW		Access file for read/write
IO.ACE		Access file for read/write/extend
	#1	File identifier pointer
	#2	Read attributes control (optional)
	#5	Access control must be enabled
IO.DAC		Deaccess file
	#1	File identifier pointer (optional)
	#2	Write attributes control list
	#5	May be nonzero but must be disabled

SUMMARY OF F11ACP FUNCTIONS

IO.EXT		Extend file
	#1	Optional if file is accessed
	#2	Placement control attribute list (optional)
	#3 & #4	Extend control The amount allocated to the file is returned in the high byte of IOST(1) plus IOST(2).
IO.RAT		Read attributes
	#1	Optional if file is accessed
	#2	Read attributes control list
IO.FNA		Find name in directory
IO.RNA		Remove name from directory
IO.ENA		Enter name in directory
	#5	May be nonzero but must be disabled
	#6	File name block pointer
IO.ULK		Unlock block
	#2	0 or count of blocks to unlock
	#4 & #5	Starting VBN to unlock or 0 to unlock all blocks.
IO.RVB		Read virtual block
IO.WVB		Write virtual block
	#1	User buffer
	#2	Buffer length
	#4 & #5	VBN

D.5 HOW TO USE THE ACP QIOS

Although the operations described in this section are normally performed by the file-access methods (RMS and FCS), your application may issue the ACP QIOs. The required parameters for each QIO are described above. The necessary steps for common operations follow.

NOTE

The file identifier is the only way to refer to a file.

D.5.1 Creating a File

To create a file:

- Use IO.CRE to create it.
- Enter it in the Master File Directory (MFD) or a user directory with IO.ENA.

D.5.2 Opening a File

To open a file:

- Use IO.FNA to find the File Identifier of the directory in the MFD.
- Use IO.FNA to find the File Identifier of the file in the directory.
- Access the file with IO.ACR, IO.ACW, or IO.ACE.

D.5.3 Closing a File

To close a file:

- Deaccess the file with IO.DAC.

D.5.4 Extending a File

To extend a file:

- Use IO.FNA to find the file identifier if the file is not accessed.
- Use IO.EXT to extend the file.

D.5.5 Deleting a File

To delete a file:

- Use IO.FNA to find the file identifier.
- Use IO.RNA to remove the directory name.
- Use IO.DEL to delete the file.

D.6 FILE HEADER BLOCK FORMAT

Table D-2 shows the format of the file header block. The various areas within the file header block are described in detail in the following sections. The offset names in the file header block may be defined either locally or globally, as shown in the following statements:

```
FHDOF$  DEF$L           ;DEFINE OFFSETS LOCALLY.

FHDOF$  DEF$G           ;DEFINE OFFSETS GLOBALLY.
```

Table D-2: File Header Block

Area	**Size (in Bytes)**	**Content**	**Offset**
Header Area	1	Identification area offset in words	H.IDOF
	1	Map area offset in words	H.MPOF
	2	File number	H.FNUM
	2	File sequence number	H.FSEQ
		number	
	-	Offset to file owner information, consisting of member number and group number	H.FOWN
	1	Member number	H.PROG

FILE HEADER BLOCK FORMAT

	1	Group number	H.PROJ
	2	File protection code	H.FPRO
	1	User-controlled file characteristics	H.UCHA
	1	System-controlled file characteristics	H.SCHA
	32.	User file attributes	H.UFAT
	-	Size in bytes of header area of file header block	S.HDHD
Identification Area	6	File name (Radix-50)	I.FNAM
	2	File type (Radix-50)	I.FTYP
	2	File version number (binary)	I.FVER
	2	Revision number	I.RVNO
	7	Revision date	I.RVDT
	6	Revision time	I.RVTI
	7	Creation date	I.CRDT
	6	Creation time	I.CRTI
	7	Expiration date	I.EXDT
	1	To round up to word boundary	
	-	Size (in bytes) of identification area of file header block	S.IDHD
Map Area	1	Extension segment number	M.ESQN
	1	Extension relative volume number (not implemented)	M.ERVN
	2	Extension file number	M.EFNU
	2	Extension file sequence	M.EFSQ

		number	
	1	Size (in bytes) of the block count field of a retrieval pointer (1 or 2); only 1 is used	M.CTSZ
	1	Size (in bytes) of the logical block number field of a retrieval pointer (2, 3, or 4); only 3 is used	M.LBSZ
	1	Words of retrieval pointers in use in the map area	M.USE
	1	Maximum number of words of retrieval pointers available in the map area	M.MAX
	-	Start of retrieval pointers	M.RTRV
	-	Size in bytes of map area of file header block	S.MPHD
Checksum Word	2	Checksum of words 0 through 255	H.CKSM

NOTE

The checksum word is the last word of the file header block. Retrieval pointers occupy the space from the end of the map area to the checksum word.

D.6.1 Header Area

The information in the header area of the file header block consists of the following:

Identification area offset - Word 0, Bits 0-7. This byte locates the start of the identification area relative to the start of the file header block. This offset contains the number of words from the start of the header to the identification area.

Map area offset - Word 0, Bits 8-15. This byte locates the start of the map area relative to the start of the file header block. This offset contains the number of words from the start of the header area to the map area.

File number - The file number defines the position this file header block occupies in the index file; for example, the index file is number 1, the storage bit map is file number 2, and so forth.

File sequence number - The file number and the file sequence number constitute the file identification number used by the system. This number is different each time a header is reused.

Structure level - This word identifies the system that created the file and indicates the file structure. A value of [1,1] is associated with all current FILES-11 volumes.

File owner information - This word contains the group number and owner number constituting the User Identification Code (UIC) for the file. Legal UICs are within the range [1,1] to [377,377]. UIC [1,1] is reserved for the system.

File protection code - This word specifies the manner in which the file can be used and who can use it. When creating the file, you specify the extent of protection desired for the file.

File characteristics - This word, consisting of two bytes, defines the status of the file.

Byte 0 defines the user-controlled characteristics, as follows:

UC.CON = 200 - Logically contiguous file. When the file is extended (for example, by a WRITE or PUT), bit UC.CON is cleared whether or not the extension requests contiguous blocks.

UC.DLK = 100 - File improperly closed.

Byte 1 defines system-controlled characteristics, as follows:

SC.MDL = 200 - File marked for delete

SC.BAD = 100 - Bad data block in file

User file attributes - This area consists of 16 words. The first seven words of this area are a direct image of the first seven words of the FDB when the file is opened. The other nine words of the record I/O control area are not used by FCS, although RMS does use them.

D.6.2 Identification Area

The information in the identification area of the file header block consists of the following:

File name - The file's creator specifies a file name of up to nine Radix-50 characters in length. This name is placed in the name field. The unused portion of the field (if any) is zero-filled.

File type - This word contains the file type in Radix-50 format.

File version number - This word contains the file version number, in binary, as specified by the creator of the file.

Revision number - This word is initialized to 0 when the file is created; it is incremented each time a file is closed after being updated or modified.

Revision date - Seven bytes are used to maintain the date on which the file was last revised. The revision date is kept in ASCII form in the format day, month, year (two bytes, three bytes, and two bytes, respectively). This date is meaningful only if the revision number is a nonzero value.

Revision time - Six bytes are used to record the time at which the file was last revised. This information is recorded in ASCII form in the format hour, minute, and second (two bytes each).

Creation date - The date on which the file was created is kept in a 7-byte field having the same format as that of the revision date (see above).

Creation time - The time of the file's creation is maintained in a 6-byte field having the same format as that of the revision time (see above).

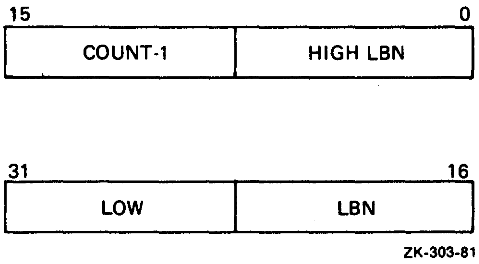
Expiration date - The date on which the file becomes eligible to be deleted is kept in a 7-byte field having the same format as that of the revision date (see above). Use of expiration is not implemented.

D.6.3 Map Area

The map area contains the information necessary to map virtual block numbers to logical block numbers. This is done by means of pointers, each of which points to an area of contiguous blocks. A pointer consists of a count field and a number field. The count field defines the number of blocks contained in the contiguous area pointed to, and the logical block number (LBN) field defines the block number of the first logical block in the area.

A value of n in the count field (following) means that n+1 blocks are allocated, starting at the specified block number.

The retrieval pointer format used in the FILES-11 file structure is as follows:



The map area normally has space for 102 retrieval pointers. It can map up to 102 discontiguous segments or up to 26112 blocks if the file is contiguous. If more retrieval pointers are required because the file is too large or consists of too many discontiguous segments, extension headers are allocated to hold additional retrieval pointers. Extension headers are allocated within the index file. They are identified by a file number and a file sequence as are other file headers; however, extension file headers do not appear in any directory.

A nonzero value in the extension file number field of the map area indicates that an extension header exists. The extension header is identified by the extension file number and the extension file sequence number. The extension segment number is used to number the headers of the file sequentially, starting with a 0 for the first.

Extension headers of a file contain a header area and identification area that are a copy of the first header as it appeared when the first extension was created. Extension headers are not updated when the first header of the file is modified.

Extension headers are created and handled by the file control primitives as needed; their use is transparent to you.

D.7 STATISTICS BLOCK

The format of the statistics block is shown in Table D-3 below. The statistics block is allocated manually in your program.

Table D-3: Statistics Block Format

Word 0	HIGH LOGICAL BLOCK NUMBER (0 if file is noncontiguous)
Word 1	LOW LOGICAL BLOCK NUMBER (0 if file is noncontiguous)
Word 2	SIZE (high)
Word 3	SIZE (low)
Word 4	LOCK COUNT ACCESS COUNT

D.8 ERRORS RETURNED BY THE FILE PROCESSORS

The error codes returned by F11ACP and MTAACP are shown in Table D-4.

Table D-4: File Processor Error Codes

ERRORS RETURNED BY THE FILE PROCESSORS

Error Code	Operations	Explanation
IE.ABO	IO.RVB/IO.WVB	Indicates that all requested data was not transferred by the device.
IE.ALC	Extend or create operation	Indicates that the operation failed to allocate the file based on placement control or because of other related problems.
IE.ALN	An attempt to access a file	Indicates that a file is already accessed on that LUN.
IE.BAD	Any function	Indicates that a required parameter is missing, that a parameter that must not be present is present, that a parameter that must be disabled is enabled, or that a parameter value is invalid.
IE.BDR	Directory operations	Indicates that you attempted a directory operation on a file that is not a directory, or that the specified directory is corrupted. This is usually caused by a 0 version number field.
IE.BHD	Any operation	Indicates that a corrupt file header was encountered, or that the operation required a feature not supported by the FCP (such as multiheader support or support for unimplemented features).
IE.BVR	Directory operations	Indicates that you attempted to enter a name

		in a directory with a negative or 0 version number.
IE.BYT	Any function	This error is returned if the buffer specified is on an odd byte boundary or is not a multiple of four bytes.
IE.BTP	Unlabeled Magtape Create	An attempt was made to create an unlabeled tape file with a record type other than fixed.
IE.CKS	Any operation	Indicates that the checksum of a file header is incorrect.
IE.CLO	File access operations	Indicates that the file was locked against access by the "deaccess lock bit."
IE.DFU	An allocation request	Indicates that there is insufficient free disk space for the requested allocation.
IE.DUP	An enter name operation	Indicates that the name and version already exist.
IE.EOF	IO.RVB/IO.WVB/IO.DEL	On read operations, this indicates an attempt to read beyond end of file. On truncate operations, it indicates an attempt to truncate a file to a length longer than that allocated or that the file was already at EOF.
IE.HFU	An extended operation	Indicates that the file header is full and cannot contain any more retrieval pointers and that adding an extension header is not allowed. When this is returned on a create operation, it

		indicates that the index file could not be extended to allow a file header to be allocated.
IE.IFC	Returned by exec	Illegal function code.
IE.IFU	Create or extend operation	Indicates that there are no file headers available based on the parameters specified when the volume was initialized.
IE.LCK	Returned on file access, directory operations, and on truncate	Indicates that the file is already accessed by a writer and that shared write has not been requested or is not allowed.
IE.LUN	Any operation requiring a file ID	Indicates that file ID has not been supplied and that the file is not accessed on the LUN.
IE.NOD	All file operations that require DSR	Indicates that an I/O request failed due to IE.UPN, that the FCP was unable to allocate required space from DSR or from secondary pool for data structures.
IE.NSF	All file operations	Indicates that the specified directory entry does not exist, that a file corresponding to the file ID does not exist, or that the file is marked for delete.
IE.OFL	Returned by exec	The device is off line.
IE.PRI	Any operation	Indicates that the user does not have the required privilege for the requested operation, or that the user has not requested the proper access to the file if the file is already accessed

		(for example, an attempt to write to a file that is accessed for read). This also indicates an attempt to do file I/O to a device that is not mounted.
IE.RER	Any operation	Indicates that the FCP encountered a fatal device read error during an operation; the operation has been aborted.
IE.SNC	Any operation	Indicates that the file number and the value contained in the header do not agree. This generally means that the header has gone bad due to a crash or a hardware error.
IE.SPC	Returned by exec	Indicates an illegal buffer.
IE.SQC	Any operation	Indicates that the file sequence number does not agree with the file header; usually indicates that the file has been deleted and the header has been reused.
IE.WAC	File access operations	Indicates that the file is already write accessed and lock against writers is requested.
IE.WAT	Write attributes and deaccess	Indicates that the FCP encountered an invalid attribute.
IE.WER	Any operation	Indicates that the FCP encountered a fatal device write error during an operation. The operation has been aborted but the disk structure may have been

		corrupted.
IE.WLK	Any operation requiring write access	Indicates that the volume is software write-locked.

D.9 FILENAME BLOCK

The format of a filename block is illustrated in Figure D-1. The offsets within the filename block are described in Table D-5.

The offset names in a filename block may be defined either locally or globally, as follows:

```
NBOF$L                  ;DEFINE OFFSETS LOCALLY.

NBOFF$  DEF$L           ;DEFINE OFFSETS LOCALLY.

NBOFF$  DEF$G           ;DEFINE OFFSETS GLOBALLY.
```

NOTE

> When you are referring to filename block locations, it is essential to use the symbolic offset names, rather than the actual addresses of such locations. The position of information within the filename block may be subject to change from release to release, whereas the offset names remain constant.

Table D-5: Filename Block Offset Definitions

Offset	Size (in Bytes)	Contents
N.FID	6	File identification field
N.FNAM	6	File name field; specified as nine characters that are stored in Radix-50 format
N.FTYP	2	File type field; specified as three characters that are stored in Radix-50 format

FILENAME BLOCK

N.FVER	2	File version number field (binary)
N.STAT	2	Filename block status word (See bit definitions in Table D-6.)
N.NEXT	2	Context for next .FIND operation
N.DID	6	Directory identification field
N.DVNM	2	ASCII device name field
N.UNIT	2	Unit number field (binary)

The bit definitions of the filename block status word (N.STAT) in the FDB and their significance are described in Table D-6.

Symbols marked with an asterisk (*) in Table D-6 indicate bits that are set if the associated information is supplied through an ASCII dataset descriptor.

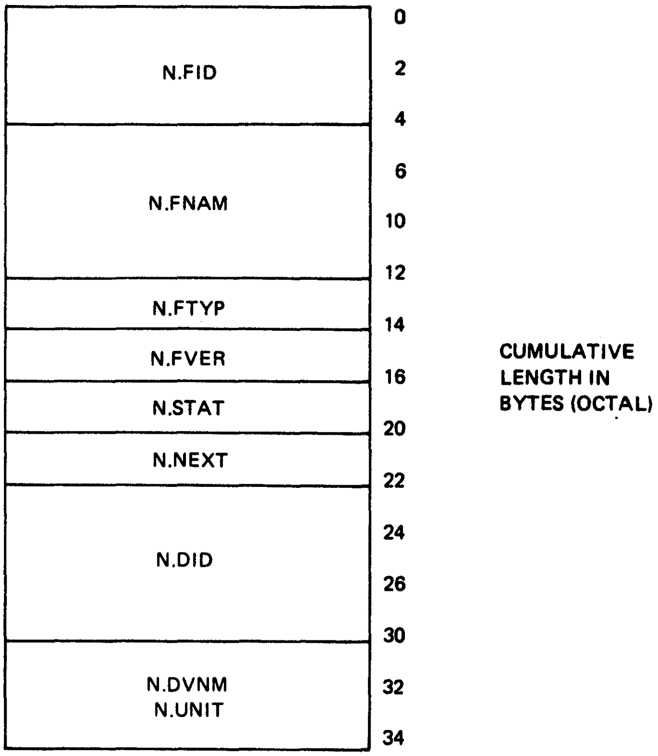


Figure D-1: Filename Block Format

Table D-6: Filename Block Status Word (N.STAT)

Symbol	Value (in Octal)	Meaning
NB.VER*	1	Set if explicit file version number is specified
NB.TYP*	2	Set if explicit file type is specified
NB.NAM*	4	Set if explicit file name is specified
NB.SVR	10	Set if wildcard file version number is specified
NB.STP	20	Set if wildcard file type is specified
NB.SNM	40	Set if wildcard file name is specified
NB.DIR*	100	Set if explicit directory string (UIC) is specified
NB.DEV*	200	Set if explicit device name string is specified
NB.SD1	400	Set if group portion of UIC contains wildcard specification*
NB.SD2	1000	Set if owner portion of UIC contains wildcard specification*
NB.ANS	2000	Set if file name is in ANSI format.

1. Although NB.SD1 and NB.SD2 are defined, they are not set or supported by FCS.

Table D-7: Filename Block Offset Definitions for ANSI Magnetic Tape

Offset	Size (in Bytes)	Definition

FILENAME BLOCK

N.FID	2	File identification field
N.ANM1	12	First 12 bytes of ANSI filename string
N.FVER	2	File version number field (binary)
N.STAT	2	Filename block status word (See bit definitions in Table D-6.
N.NEXT	2	Context for next .FIND operation
N.ANM2	6	Remainder of the ANSI file name string
N.DVNM	2	ASCII device name field
N.UNIT	2	Unit number field (binary)

The bit definitions of the filename block status word (N.STAT) are shown in Table D-6.

APPENDIX E

QUAD SERIAL LINE UNIT (PC3XC-BA)

E.1 INTRODUCTION

The Quad Asynchronous Communications Module (PC3XC-BA) is a Professional 300 series module that provides four full-duplex, serial asynchronous ports. The ports are EIA RS232C/RS-423A compatible with user selectable baud rates from 50 to 38.4K baud, contingent on system software. Each receiver is buffered four times.

The ports are accessed through an external connector box containing four 25-pin D-sub connectors. The connector box is connected to the module by a 40-conductor ribbon cable.

The four ports have modem control which can be configured in two ways :

- Two ports with full asynchronous modem support and two ports with no modem support
- Four ports with partial modem support

NOTE

> The P/OS terminal driver, which supports the Quad Serial Line Unit, does not support the use of modems or the modem control signals.

The configuration is determined by the connector box.

The PC3XC-BA hardware consists of the controller module, a ribbon cable, and a D-sub adapter module in a connector box.

E.2 THEORY OF OPERATION

The four-serial-line module consists of the following sections:

- Two Dual Universal Asynchronous Receiver/Transmitters (DUARTs)
- Interrupt logic
- Modem control
- Diagnostic ROM
- CTI bus interface
- Connecter adapter box

E.2.1 DUARTs

Two DUARTs are used in the four-serial-line module. Each DUART provides two independent channels. Each channel has several unique registers (i.e., Mode Registers, Status Registers) as well as several registers that are shared between the two channels of the DUARTs (i.e., Interrupt Mask Register, Interrupt Status Register). All registers hold eight bits of data.

Each channel of each DUART has a full-duplex asynchronous receiver/transmitter. The operating frequency for each receiver and transmitter can be selected independently.

The transmitter accepts parallel data from the CPU, converts it to serial bit stream, inserts the appropriate start, stop, and optional parity bits and outputs a composite serial stream of data. The receiver accepts serial data, converts this serial input to parallel format, checks for start bit, stop bit, parity bit (if any), or break condition and sends an assembled character to the CPU.

Each receiver is capable of holding up to four characters (a three character FIFO plus a character in the Receive Holding Register) before a data overrun occurs.

E.2.2 Interrupt Logic

The four-serial-line module uses only the option module interrupt request A to interrupt the CPU. Interrupt requested by either

DUART causes an interrupt through the option module interrupt request A vector. Interrupts to the CPU only occur if interrupts are enabled in the Module Status Register (MSR).

The Module Status Register provides an indication of any interrupts pending and the state of the interrupt enable bit. Interrupts are enabled/disabled by setting/clearing the interrupt enable bit. This allows the interrupt request to be re-toggled in the event an interrupt occurs before the previous interrupt has been serviced. The Module Status Register also identifies which DUART is requesting the interrupt.

The events which can cause an interrupt are selectable by writing to the appropriate Interrupt Mask Register for the corresponding channel. The following events can be programmed to cause an interrupt:

- A transmitter is ready to transmit a character.
- A character has been received, or three characters have been received (FIFO full).
- A receiver has detected the beginning or the end of a received break.

Interrupt controllers on the PC350 and PC380 are edge triggered. Interrupts in the DUARTs and on the quad module are ORed. This dissimilarity must be handled in software.

The problem is that when two (or more) interrupt conditions occur, only one interrupt will be generated (one IRQA). IRQA does not deassert when a particular interrupt condition is serviced if another interrupt condition is pending.

In the interrupt service routine, one of the first instructions should disable interrupts (bit 6 of MSR) from the bus. This forces deassertion of IRQA. Psuedo-code follows:

```
Disable module interrupts          ; bit 6 of BASE + 4
Do while interrupt pending         ; bit 7 of BASE + 4
Determine interrupt cause
  Service interrupt
End do
Enable module interrupts           ; bit 6 of BASE + 4
```

E.2.3 Modem Controls

There are two possible modem configurations, determined by the connector box.

- Configuration 2/2 - Two ports with full asynchronous modem control and two with no modem control

  Channel 0

  - Inputs : CTS, DSR, CD, RI, TI, SPDMI
  - Outputs: RTS, LL, DTR, RL, DSRS

  Channel 1

  - No modem controls

  Channel 2

  - Inputs : CTS, DSR, CD, RI, TI, SPDMI
  - Outputs: RTS, LL, DTR, RL, DSRS

  Channel 3

  - No modem controls

- Configuration 4/0 - Four ports with partial asynchronous modem control

  Channels 0,1,2,3

  - Inputs : CTS, CD, DSR
  - Outputs: RTS, DTR

E.3 DUART TRANSMITTER OPERATION

When the transmitter is enabled through the command register, the DUART indicates to the CPU that it is ready to accept data by setting the transmitter-ready bit in the status register. This condition can be programmed to generate an interrupt request. When a character is loaded into the transmit holding register

(THR), transmitter ready is cleared. Data is transferred from the THR to the transmit shift register when it is idle or has completed transmission of the previous character. Transmitter ready is then reasserted. Characters cannot be loaded into the THR while the transmitter is disabled.

The transmitter converts the parallel data from the CPU to a serial bit stream on the transmit data output pin. It automatically sends a start bit followed by the programmed number of data bits, an optional parity bit, and the programmed number of stop bits. The least significant bit is sent first. Following the transmission of the stop bits, if a new character is not available in the THR, the transmit data output remains high and the transmit-then-empty status bit is set to 1. Transmission resumes and the transmit-then-empty bit is cleared when the CPU loads a new character into the THR. If the transmitter is disabled, it continues operating until the character currently being transmitted is completely sent out. The transmitter can be forced to send a continuous low condition by issuing a send break command.

The transmitter can be reset though a software command. If it is reset, operation ceases immediately and the transmitter must be enabled through the command register before resuming operation. If CTS operation is enabled, the clear-to-send input must be low in order for the character to be transmitted. If it goes high in the middle of a transmission, the character in the shift register is transmitted and transmit data output then remains in the marking state until the clear-to-send bit goes low. The transmitter can also control the deactivation of the request-to-send output. If programmed, the request-to-send output will be reset one bit time after the character in the transmit shift register and THR (if any) are completely transmitted, if the transmitter has been disabled.

E.3.1 Receiver Operation

The DUART is conditioned to receive data when enabled through the command register. The receiver looks for a high to low (mark to space) transition of the start bit on the receive data input pin. If a transition is detected, the state of the receive data input is sampled each 16X clock for 7-1/2 clocks (16X clock mode) or at the next rising edge of the bit time clock (1X clock mode). If receive data input is sampled high, the start bit is invalid and the search for a valid start bit begins again. If receive data input is still low, a valid start bit is assumed and the receiver continues to sample the input at one bit time intervals at the theoretical center of the bit, until the proper number of data bits and the parity bit (if any) have been assembled, and one stop bit has been detected. The least significant bit is received first. The data is then transferred to the Receiver Holding Register (RHR), and the receive-ready bit in the Status Register (SR) is set to '1'. This condition can be programmed to generate an interrupt. If the character length is less than eight bits, the most significant unused bits in the RHR are set to zero.

After the stop bit is detected, the receiver will immediately look for the next start bit. However if a non-zero character was received without a stop bit (framing error) and receive data input remains low for one-half of the bit period after the stop bit was sampled, then the receiver operates as if a new start bit transition had been detected at that point (one-half bit time after the stop bit was sampled).

The parity error, framing error, overrun error and received break state (if any) are strobed into the SR at the received character boundary, before the receiver-ready status bit is set. If a break condition is detected (receive data input is low for the entire character including the stop bit), a character consisting of all zeros will be loaded into the RHR and the received break bit in the SR is set to one. The receive data input must return to a high condition for at least one-half bit time before a search for the next start bit begins.

The RHR consists of a first-in-first-out (FIFO) stack with a capacity of three characters. Data is loaded from the receive shift register into the topmost empty position of the FIFO. The receiver-ready bit in the status register is set whenever one or more characters are available to be read, and a FFULL status bit is set if all three stack positions are filled with data. Either of these bits can be selected to cause an interrupt. A read of the RHR outputs the data at the top of the FIFO. After the read cycle, the data FIFO and its associated status bits are 'popped' thus emptying a FIFO position for new data.

In addition to the data word, three status bits (parity error, framing error, and received break) are also appended to each data character in the FIFO(overrun is not). Status can be provided in two ways, as programmed by the error mode control bit in the mode register. In the 'character' mode, the status applies only to the character at the top of the FIFO. In the 'block' mode, the status provided in the SR for these three bits is the logical OR of the status for all characters coming to the top of the FIFO since the last 'reset error' command was issued. In either mode, reading the SR does not affect the FIFO. The FIFO is popped only when the RHR is read. Therefore the status register should be read prior to reading the FIFO.

If the FIFO is full when a new character is received, that character is held in the receive shift register until a FIFO position is available. If an additional character is received, the contents of the FIFO are not affected - the character in the receive shift register is lost and the overrun error status bit is set upon receipt of the start bit of the new (overrunning) character.

The receiver can control the deactivation of request to send. If programmed to operate in this mode, the request-to-send output will be negated when a valid start bit was received and the FIFO is full. When a FIFO position becomes available, the request-to-send output will be re-asserted automatically. Ths feature can be used to prevent an overrun, in the receiver, by connecting the request-to-send output to the clear-to-send input of the transmitting device.

If the receiver is disabled, the FIFO characters can be read. However, no additional characters can be received until the receiver is enabled again. If the receiver is reset, the FIFO and all of the receiver status, and the corresponding output ports and interrupt are reset. No additional characters can be received until the receiver is enabled again.

E.3.2 Interrupts

Each DUART provides a single active low interrupt output, which is activated upon the occurrence of any of eight internal events. Associated with the interrupt system are the interrupt mask register (IMR) and the interrupt status register (ISR). The IMR may be programmed to select only certain conditions to cause an interrupt. The ISR can be read by the CPU to determine all currently active interrupting conditions.

E.3.3 Programming

The contents of certain control registers are initialized to zero on power-up. Operational problems may result if certain register contents are changed during operation. For example, changing the number of bits per character while the transmitter is active may cause the transmission of an incorrect character. In general, the contents of the MR, CSR, and OPCR should only be changed while the receivers and transmitters are not enabled.

Mode registers 1 and 2 of each channel are accessed with independent auxiliary pointers. The pointer is set to MR1 with a 'reset pointer' command. Any read or write of the mode register while the pointer is at MR1 switches the pointer to MR2. The pointer then remains at MR2, so that subsequent accesses are always to MR2 unless the 'reset pointer' command is issued.

Mode, command, clock select, and status registers are duplicated for each channel to provide total independent operation and control.

E.3.4 Multidrop Mode

The DUART is equipped with a wake-up mode used for multidrop applications. This mode is selected by programming bits MR1A[4:3] or MR1B[4:3]to '11' for channels A and B. In this mode of operation, a 'master' station transmits an address character followed by data characters for the addressed 'slave' station. The slave stations, with receivers that are normally disabled, examine the received data stream and 'wake-up' the CPU (by setting receiver ready) only upon receipt of an address character. The CPU compares the received address to its station address and enables the receiver if it wishes to receive the subsequent data characters. Upon receipt of another address character, the CPU may disable the receiver to initiate the process again.

A transmitted character consists of a start bit, the programmed number of data bits, an address/data (A/D) bit, and the programmed number of stop bits. The polarity of the transmitted A/D bit is selected by the CPU by programming bit MR1[2]. MR1[2] = 0 transmits a zero in the A/D bit position, which identifies the corresponding bits as data. MR1[2] = 1 transmits a one in the A/D bit position, which identifies the corresponding bits as an address. The CPU should program the mode register prior to loading the corresponding data bits into the THR.

In this mode, the receiver continuously looks at the received data stream, whether it is enabled or disabled. If disabled, it sets the receiver ready status bit and loads the character into the RHR FIFO if the received A/D bit is a one (address tag), but discards the received character if the received A/D bit is a zero (data tag). If enabled, all received characters are tranferred to the CPU via the RHR. In either case, the data bits are loaded into the data FIFO while the A/D bit is loaded into the status FIFO position normally used for parity error. Framing error, overrun error, and break detect operate normally whether or not the receiver is enabled.

E.3.5 Registers

Figure E-1 shows the addressing scheme for PRO 300 Modules.

```
+---+-------+-------+-------+-------+-------+-------+-------+
| 1 | 1 1 1 | 1 1 1 | 1 1 1 | 1 0 S | S S X | X X X | X X X |
+---+-------+-------+-------+-------+-------+-------+-------+
```

Figure E-1: PRO 300 Modules Addressing Scheme

SSS is the slot position (relative to 0) and XXXXXXX is the offset from the base address.

The registers on this module are located as shown in Table E-1.

Table E-1: PR0 300 Module Register Summary

Base +	Function
176 . . . 140	DUART 1 Registers Channels 2 and 3
136 . . . 100	DUART 0 Registers Channels 0 and 1
76 . . . 06	(RESERVED)
04	Module Status Register
02	Diagnostic ROM Address Register

Base +	**Function**
00	Diagnostic ROM Data Register

All registers are 8 bits wide. A word read will return 0s in the high byte of defined registers. Writes to the high byte of defined registers will have no effect. Reading RESERVED registers will return unpredictable data. Writing to RESERVED registers should be avoided.

E.3.6 Diagnostic ROM Data Register (RDR)

The low byte of the RDR is a data window into the Diagnostic ROM. Each time the RDR is read, the data in the ROM pointed to by the ROM address register is put on the bus, then the address pointer is incremented to the next location. The high byte is read as zeros and writes to this register have no effect.

E.3.7 Diagnostic ROM Address Register (RAR)

The RAR is used to reset the address pointer for the diagnostic ROM. Any data written to this register resets the pointer to the first byte of the ROM. Reads to this register will return zeros.

E.3.8 Module Status Register (MSR)

Figure E-2 shows the Module Status Register (MSR):

7	6	5,4	3	2	1	0
INT PENDING	INT ENABLE	Reserved	CONFIG 1	CONFIG 0	CH 2/3	CH 0/1

Figure E-2: Module Status Register

The MSR provides a simple means of monitoring and controlling interrupts on the module.

Bit 7 - Interrupt Pending

This bit is set if there is any active interrupt pending. If the interrupt enable bit is set, it will cause Interrupt Request A to be asserted on the bus. This bit is read-only.

Bit 6 - Interrupt Enable

This bit controls the IRQA line. If it is set and either DUART has an interrupt pending, then Interrupt Request A is asserted.

Bits 3,2 - Configuration bits.

Table E-2: Configuration of Bits 2 and 3 (MSR)

Config 1	**Config 2**	**Configuration**
0	0	no connector box present
0	1	2/2 - 2 channels with full asynchronous modem control, 2 channels with no modem control
1	0	4/0 - 4 channels with partial asynchronous modem control
1	1	manufacturing mode

Bit 1 - Set if the pending interrupt came from channel 2 or channel 3.

Bit 0 - Set if the pending interrupt came from channel 0 or channel 1.

E.3.9 DUART Registers

Table E-3 shows the DUART 0 registers. For DUART 1 add 40 to the listed address offsets.

NOTE

> In DUART descriptions, Channel A will be channel 0 in DUART 0, and Channel B will be channel 1 in DUART 0. Also, Channel A will be channel 2 in DUART 1, and Channel B will be channel 3 in DUART 1.

Table E-3: DUART 0 Register Addresses

BASE+	Register Read	Register Write
100	Mode Register A (MR1A,MR2A)	Mode Register A (MR1A,MR2A)
102	Status Register A (SRA)	Clock Select Reg. A (CSRA)
104	(RESERVED)	Command Register A (CRA)
106	RX Holding Register A (RHRA)	TX Holding Register A (THRA)
110	Input Port Change Reg. (IPCR)	Aux. Control Reg. (ACR)
112	Interrupt Status Reg. (ISR)	Interrupt Mask Reg. (IMR)
114	Counter/Timer Upper (CTU)	C/T Upper Register (CTUR)

BASE+	**Register Read**	**Register Write**
116	Counter/timer Lower (CTL)	C/T Lower Register (CTLR)
120	Mode Register B (MR1B,MR2B)	Mode Register B (MR1B,MR2B)
122	Status Register B (SRB)	Clock Select Reg. B (CSRB)
124	(RESERVED)	Command Register B (CRB)
126	RX Holding Register B (RHRB)	TX Holding Register B (THRB)
130	(RESERVED)	(RESERVED)
132	Input Port	Output Port Conf. Reg. (OPCR)
134	Start Counter Command	Set Output Port Bits Command
136	Stop Counter Command	Reset Output Port Bits Command

E.3.9.1 Channel A Mode Register 1 (MR1A) -

READ/WRITE
Channel 0 address BASE + 100
Channel 2 address BASE + 140

MR1A is accessed when the Channel A MR pointer points to MR1. The pointer is set to MR1 by RESET or by a "set pointer" command applied via CRA. After reading or writing MR1A, the pointer will point to MR2A. Figure E-3 shows Channel A Mode Register 1

(MR1A).

```
     7         6          5          4-3          2            1-0
--------------------------------------------------------------------
|    RX    |    RX    |          |          |          |
| RTS CTL  | INT SEL  | ERR MODE | PAR MODE | PAR TYPE | BITS PER CHA
--------------------------------------------------------------------
```

Figure E-3: Channel A Mode Register 1 (MR1A)

Bit 7 - Channel A Receiver Request-to-send Control (0 = no, 1 = yes)

This bit controls the deactivation of the RTS output by the receiver. This output is normally asserted by setting OPR[0] and negated by resetting OPR[0]. MR1A[7]=1 causes RTSAN to be negated upon receipt of a valid start bit if the Channel A FIFO is full. However, OPR[0] is not reset and RTSAN will be asserted again when an empty FIFO position is available. This feature can be used for flow control to prevent overrun in the receiver by using the RTS output signal to control the CTS input of the transmitting device.

Bit 6 - Channel A Receiver Interrupt Select (0 = RxRDY, 1 = FFULL)

This bit selects either the Channel A receiver-ready status (RXRDY) or the Channel A FIFO full status (FFULL) to be used for CPU interrupts.

Bit 5 - Channel A Error Mode Select (0 = char, 1 = block)

This bit selects the operating mode of the three FIFOed status bits (FE, PE, received break) for Channel A. In the "character" mode, status is provided on a character-by-character basis : the status applies only to the character at the top of the FIFO. In the "block" mode, the status provided in the SR for these bits is the accumulation (logical OR) of the status for all characters coming to the top of the FIFO since the last "reset error" command for Channel A was issued.

Bits 4-3 - Channel A Parity Mode Select

00	with parity
01	force parity
10	no parity

11	multidrop mode

If "with parity" or "force parity" is selected, a parity bit is added to the transmitted character and the receiver performs a parity check on incoming data. A "11" in these bits selects Channel A to operate in the special multidrop mode.

Bit 2 - Channel A Parity Type Select (0 = even, 1 = odd)
This bit selects the parity type (odd or even) if the "with parity" mode is programmed, and the polarity of the forced parity bit if the "force parity" mode is programmed. It has no effect if the "no parity" mode is programmed. In the special multidrop mode it selects the parity of the A/D bit.

Bits 1-0 - Channel A Bits Per Character Select

00	5 bits
01	6 bits
10	7 bits
11	8 bits

This field selects the number of data bits per character to be transmitted and received. The character length does not include the start, parity, and stop bits.

E.3.9.2 Channel A Mode Register 2 (MR2A) -

READ/WRITE
Channel 0 address BASE + 100
Channel 2 address BASE + 140

MR2A is accessed when the Channel A MR pointer points to MR2, which occurs after any access to MR1A. Accesses to MR2A do not change the pointer. Figure E-4 shows Channel A Mode Register 2 (MR2A).

7-6	5	4	3-0
CHANNEL MODE	TX RTS CTL	TX CTS EN	STOP BIT LENGTH

Figure E-4: Channel A Mode Register 2 (MR2A)

Bits 7-6 - Channel A Mode Select

Each channel of the DUART can operate in one of four modes:
normal (00), auto echo (01), local loop (10), or remote loop (11).

"00" - Normal Mode

Transmitter and receiver operating independently.

"01" - Automatic Echo Mode

Automatically retransmits the received data.
The following conditions are true while in echo mode:

- Received data is reclocked and retransmitted.
- The receive clock is used for the transmitter.
- The receiver must be enabled, but the transmitter need not be enabled.
- The Channel A transmitter ready and transmitter-empty status bits are inactive.
- The received parity is checked, but is not regenerated for transmission, i.e., transmitted parity bit is as received.
- Character framing is checked, but the stop bits are retransmitted as received.
- A received break is echoed as received until the next valid start bit is detected.
- CPU to receiver communication continues normally, but the CPU to transmitter link is disabled.

"10" - Local Loopback Diagnostic Mode

In this mode the following are true:

- The transmitter output is internally connected to the receiver input.
- The transmitter clock is used for the receiver.
- The transmit data output lead is held high.
- The receive data input lead is ignored.
- The transmitter must be enabled, but the receiver need not be enabled.
- CPU to transmitter and receiver communications continue normally.

"11" - Remote Loopback Diagnostic Mode

In remote loopback diagnostic mode, the following is true:

- Received data is relocked and retransmitted on the transmit data output lead.
- The receive clock is used for the transmitter.
- Received data is not sent to the local CPU, and the error status condition is inactive.
- The received parity is not checked and is not regenerated for transmission, i.e., transmitted parity bit is as received.
- The receiver must be enabled.
- Character framing is not checked, and the stop bits are retransmitted as received.
- A received break is echoed as received until the next valid start bit is detected.

The user must exercise care when switching into and out of the various modes. The selected mode will be activated immediately upon mode selection, even if this occurs in the middle of a received or transmitted character. Likewise, if a mode is deselected, the device will switch out of the mode immediately. An exception to this is switching out of autoecho or remote loopback modes : if the deselection occurs just after the receiver has sampled the stop bit (indicated in autoecho by assertion of receive ready), and the transmitter is enabled, the transmitter will remain in autoecho mode until the entire stop bit has been retransmitted.

Bit 5 - Channel A Transmitter Request-to-Send (0 = no, 1 = yes)
This bit controls the deactivation of the RTS output by the transmitter. This output is normally asserted by setting OPR[0] and negated by resetting OPR[0]. A "1" in this bit causes OPR[0] to be reset automatically one bit time after the characters in the Channel A transmit shift register and in the THR, if any, are completely transmitted, including the programmed number of stop bits, if the transmitter is not enabled. This feature can be used to automatically terminate the transmission of a message as follows :

- Program auto-reset mode (bit 5 = 1).
- Enable transmitter.
- Assert RTSAN (OPR[0] = 1).
- Send message.
- Disable transmitter after the last character is loaded into the Channel A THR.
- The last character will be transmitted and OPR[0] will be reset one bit time after the last stop bit, causing RTSAN to be negated.

Bit 4 - Channel A Clear-to-Send Control (0 = no, 1 = yes)
If this bit is zero, CTS has no effect on the transmitter. If this bit is a one, the transmitter checks the state of CTS each time it is ready to send a character. If CTS is asserted, the character is transmitted. If it is negated, the transmit data output remains in the marking state and the transmission is delayed until CTS is asserted. Changes in CTS while a character is being transmitted do not affect the transmission of that character.

Bits 3,2,1,0 - Channel A Stop Bit Length Select
This field programs the length of the stop bit appended to the transmitted character. Stop bit lengths of 9/16 to 1 and 1-9/16 to 2 bits, in increments of 1/16 bit, can be programmed for character lengths of 6,7, and 8 bits. For a character length of 5 bits, 1-1/16 to 2 bits can be programmed in increments of 1/16 bit. The receiver only checks for a "mark" condition at the center of the first stop bit position (one bit time

after the last data bit, or after the parity bit if parity is enabled) in all cases.

If an external 1X clock is used for the transmitter, bit 3 = 0 selects one stop bit and bit 3 = 1 selects two stop bits to be transmitted.

0 = 0.563

1 = 0.625

2 = 0.688

3 = 0.750

4 = 0.813

5 = 0.875

6 = 0.938

7 = 1.000

8 = 1.563

9 = 1.625

A = 1.688

B = 1.750

C = 1.813

D = 1.875

E = 1.938

F = 2.000

NOTE

Add 0.5 to values shown for 0-7 if channel is programmed for 5 bits/character.

E.3.9.3 Channel A Status Register (SRA) -

READ ONLY
Channel 0 address BASE + 102
Channel 2 address BASE + 142

Figure E-5 shows the Channel A Status Register (SRA).

7	6	5	4	3	2	1	0
REC BREAK	FRAME ERROR	PAR ERR	OVRUN ERROR	TxEMT	TxRDY	FFULL	RxRDY

Figure E-5: Channel A Status Register

Bit 7 - Channel A Received Break (0 = no, 1 = yes)
This bit indicates that an all zero character of the programmed length has been received without a stop bit.

Only a single FIFO position is occupied when a break is received - further entries to the FIFO are inhibited until the receive data line returns to the marking state for at least one-half a bit time (two successive edges of the internal or external 1X clock).

When this bit is set, the Channel A 'delta break' bit in the ISR (ISR[2]) is set. ISR[2] is also set when the end of the break condition, as defined above, is detected.

The break detect circuitry can detect breaks that originate in the middle of a received character. However, if a break begins in the middle of a character, it must persist until at least the end of the next character time in order for it to be detected.

Bit 6 - Channel A Framing Error (0 = no, 1 = yes)
This bit, when set, indicates that a stop bit was not detected when the corresponding data character in the FIFO was received. The stop bit check is made in the middle of the first stop bit position.

Bit 5 - Channel A Parity Error (0 = no, 1 = yes)
This bit is set when the 'with parity' or 'force parity' mode is programmed and the corresponding character in the FIFI was received with incorrect parity.

In the special multidrop mode the parity error bit stores the received A/D bit.

NOTE

Bits 7-5 are appended to the corresponding character in the receive FIFO. A read of this register provides bits 7-5 from the top of the FIFO. These bits are cleared by a "reset error status" command. In character mode they are discarded when the corresponding data character is read from the FIFO.

Bit 4 - Channel A Overrun Error (0 = no, 1 = yes)
This bit, when set, indicates that one or more characters in the receive data stream have been lost. It is set upon receipt of a new character when the FIFO is full and a character is already in the receive shift register waiting for an empty FIFO position. When this occurs, the character in the receive shift register (and its break detect, parity error and framing error status, if any) is lost.

This bit is cleared by a 'reset error status' command.

Bit 3 - Channel A Transmitter Empty (TxEMTA) (0 = no, 1 = yes)
This bit will be set when the Channel A transmitter underruns; i.e., both the transmit holding register (THR) and the transmit shift register are empty. It is set after transmission of the last stop bit of a character if no character is in the THR awaiting transmission. It is reset when the THR is loaded by the CPU or when the transmitter is disabled.

Bit 2 - Channel A Transmitter Ready (TxRDYA) (0 = no, 1 = yes)
This bit, when set, indicates that the THR is empty and ready to be loaded with a character. This bit is cleared when the THR is loaded by the CPU and is set when the character is transferred to the transmit shift register. Transmit ready is reset when the transmitter is disabled and is set when the transmitter is first enabled, i.e., characters loaded into the THR while the transmitter is disabled will not be transmitted.

Bit 1 - Channel A FIFO Full (FFULLA) (0 = no, 1 = yes)
This bit is set when a character is transferred from the receive shift register to the receive FIFO and the transfer causes the FIFO to become full; i.e., all three FIFO positions are occupied. It is reset when the CPU reads the RHR. If a character is waiting in the receive shift register because the FIFO is full, FFULL will not be reset when the CPU reads the RHR.

Bit 0 - Channel A Receiver Ready (RxRDYA) (0 = no, 1 = yes)
This bit indicates that a character has been received and is waiting in the FIFO to be read by the CPU. It is set when the character is transferred from the receive shift regster to the FIFO and reset whenthe CPU reads the RHR, if after this read there are no more characters in the FIFO.

E.3.9.4 Channel A Clock Select Register (CSRA) -

WRITE ONLY
Channel 0 address BASE + 102
Channel 2 address BASE + 142

Figure E-6 shows the Channel A Clock Select Register (CSRA).

```
       7,6,5,4                      3,2,1,0
----------------------------------------------------
| Receiver Clock Select   | Transmitter Clock Select |
----------------------------------------------------
```

Figure E-6: Channel A Clock Select Register (CSRA)

Bits 7,6,5,4 - Receiver Clock Select
This field selects the baud rate clock for the Channel A receiver as follows in Table E-4:

Table E-4: Channel A Receiver Baud Rate

Bits 7,6,5,4	**ACR[7]=0**	**Baud Rate ACR[7]=1**
0 0 0 0	50	75
0 0 0 1	110	110
0 0 1 0	134.5	134.5
0 0 1 1	200	150
0 1 0 0	300	300
0 1 0 1	600	600
0 1 1 0	1200	1200
0 1 1 1	1050	2000
1 0 0 0	2400	2400
1 0 0 1	4800	4800

Bits 7,6,5,4	**ACR[7]=0**	**Baud Rate ACR[7]=1**
1 0 1 0	7200	1800
1 0 1 1	9600	9600
1 1 0 0	38400	19200
1 1 0 1	timer	timer
1 1 1 0	DO NOT USE	
1 1 1 1	DO NOT USE	

Bits 3,2,1,0 - Channel A Transmitter Clock Select
This field selects the baud rate for the Channel A transmitter. Definitions are the same as for the Channel A receiver.

E.3.9.5 Channel A Command Register (CRA) -

WRITE ONLY
Channel 0 address BASE + 104
Channel 2 address BASE + 144

CRA is a register used to supply commands to Channel A. Multiple commands can be specified in a single write to CRA as long as the commands are non-conflicting (for example, the "enable transmitter" and "reset transmitter" commands cannot be specified in a single command word).

Figure E-7 shows the Channel A Command Register (CRA).

7	6-4	3	2	1	0
	misc comm	dis Tx	en Tx	dis Rx	en Rx

Figure E-7: Channel A Command Register

Bit 7 - Bit 7 is not used; it must be zero.

Bits 6-4 - miscellaneous commands - these bits may be used to specify a single command as follows:

0 0 0	No command
0 0 1	Reset MR pointer. Causes the Channel A MR pointer to point to MR1.
0 1 0	Reset receiver. Resets the Channel A receiver as if a hardware reset had been applied. The receiver is disabled and the FIFO is flushed.
0 1 1	Reset transmitter. Resets the Channel A transmitter as if a hardware reset had been applied.
1 0 0	Reset error status. Clears the Channel A received break, parity error, framing error, and overrun bits in the status register (SRA[7:4]). Used in character mode to clear OE status (SRA[7:4] are also cleared) and in block mode to clear all error status after a block of data has been received.
1 0 1	Reset break change interrupt. Causes the Channel A break detect change bit in the interrupt status register (ISR[2]) to be cleared to zero.
1 1 0	Start break. Forces the TDXA output low (spacing). If the transmitter is empty the start of the break condition will be delayed up to two bit times. If the transmitter is active the break begins when transmission of the character is completed. If a character is in the THR, the start of the break will be delayed until that character or any others loaded subsequently are transmitted. The transmitter must be enabled for this command to be accepted.
1 1 1	Stop break. The TXDA line will go high (marking) within two bit times. TXDA will remain high for one bit

time before the next character, if any, is transmitted.

Bit 3 - Disable Channel A transmitter (0 = no, 1 = yes)
This command terminates transmitter operation and resets the transmitter-ready and transmitter-empty status bits. However, if a character is being transmitted or if a character is in the THR when the transmitter is disabled, the transmission of the character(s) is completed before assuming the inactive state.

Bit 2 - Enable Channel A transmitter (0 = no, 1 = yes)
Enables operation of the Channel A transmitter. The transmitter-ready bit will be asserted.

Bit 1 - Disable Channel A receiver (0 = no, 1 = yes)
The command terminates operation of the receiver immediately - a character being received will be lost. The command has no effect on the receiver status bits or any other control registers. If the special multidrop mode is programmed, the receiver operates even if it is disabled.

Bit 0 - Enable Channel A receiver (0 = no, 1 = yes)
Enables operation of the Channel A receiver. If not in the special wake-up mode, this also forces the receiver into the search for start-bit state.

E.3.9.6 Channel A Receive Holding Register (RHRA) -

READ ONLY
Channel 0 address BASE + 106
Channel 2 address BASE + 146

The RHR consists of a first-in-first-out (FIFO) stack with a capacity of three characters. Data is loaded from the receive shift register into the topmost empty position of the FIFO. The receiver-ready bit in the status register is set whenever one or more characters are available to be read, and a FFULL status bit is set if all three stack positions are filled with data. A read of the RHR outputs the data at the top of the FIFO. After the read cycle, the data FIFO and its associated status bits are 'popped' thus emptying a FIFO position for new data.

In addition to the data word, three status bits (parity error, framing error, and received break) are also appended to each data character in the FIFO. In the 'character' mode, the status applies only to the character at the top of the FIFO. In the 'block' mode, the status provided in the SR for these three bits is the logical OR of the status for all characters coming to the top of the FIFO since the last 'reset error' command was issued. The FIFO is popped only when the RHR is read. Therefore the status register should be read prior to reading the FIFO.

If the FIFO is full when a new character is received, that character is held in the receive shift register until a FIFO position is available. If an additional character is received, the contents of the FIFO are not affected - the character in the receive shift register is lost and the overrun error status bit is set.

If the receiver is disabled, the FIFO characters can be read. However, no additional characters can be received until the receiver is enabled again. If the receiver is reset, the FIFO and all of the receiver status, and the corresponding output ports and interrupt are reset. No additional characters can be received until the receiver is enabled again.

E.3.9.7 Channel A Transmitter Holding Register (THRA) -

WRITE ONLY
Channel 0 address BASE + 106
Channel 2 address BASE + 146

When the transmitter is enabled through the command register, the transmitter-ready bit gets set in the status register. When a character is loaded into the transmit holding register (THR), transmitter ready gets cleared. Data is transferred from the THR to the transmit shift register when it is idle or has completed transmission of the previous character. Transmitter-ready is then reasserted. Characters cannot be loaded into the THR while the transmitter is disabled.

The transmitter converts the parallel data from the CPU to a serial bit stream on the transmit data output pin. It automatically sends a start bit followed by the programmed number of data bits, an optional parity bit, and the programmed number of stop bits. The least significant bit is sent first. Following the transmission of the stop bits, if a new character is not available in the THR, the transmit data output remains high and the transmitter-empty status bit is set to 1. Transmission resumes and the transmitter empty bit is cleared when the CPU loads a new character into the THR. If the transmitter is disabled, it continues operating until the character currently being transmitted is completely sent out. The transmitter can be forced to send a continuous low condition by issuing a send break command.

DUART TRANSMITTER OPERATION

The transmitter can be reset though a software command. If it is reset, operation ceases immediately and the transmitter must be enabled through the command register before resuming operation.

E.3.9.8 Input Port Change Register (IPCR) -

READ ONLY
DUART 0 address BASE + 110
DUART 1 address BASE + 150

Bits 7,6,5,4 - IP3, IP2, IP1, IP0 Change of State (0 = no, 1 = yes)

These bits are set when a change of state occurs at the respective input pins. They are cleared when the IPCR is read by the CPU. A read of the IPCR also clears ISR[7], the input change bit in the interrupt status register. The setting of these bits can be programmed to generate an interrupt to the CPU.

Bits 3,2,1,0 - IP3, IP2, IP1, IP0 Current State (0 = low, 1 = high)

These bits provide the current state of the respective inputs. The information is unlatched and reflects the state of the input pins at the time the IPCR is read.

E.3.9.9 Auxiliary Control Register (ACR) -

WRITE ONLY
DUART 0 address BASE + 110
DUART 1 address BASE + 150

Figure E-8 shows the Auxiliary Control Register (ACR).

7	6,5,4	3	2	1	0
Baud Rate Generator Set Select	CNT/TIME Mode and Source	delta IP3 int	delta IP2 int	delta IP1 int	delta IP0 int

Figure E-8: Auxiliary Control Register

Bit 7 - Baud Rate Generator Set Select

This bit selects one of two sets of baud rates to be generated by the baud rate generator.

- Bit 7 = 0 selects : 50, 110, 134.5, 200, 300, 600, 1050, 1200, 2400, 4800, 7200, 9600, 38400
- Bit 7 = 1 selects : 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2000, 2400, 4800, 9600, 19200

Table E-5 shows baud rate generator characteristics.

Table E-5: Baud Rate Generator Characteristics (CLOCK = 3.6864 MHZ)

Nominal Baud Rate	**Actual 16x Clock (KHZ)**	**Error (percent)**
50	0.8	0
75	1.2	0
110	1.759	-0.069
134.5	2.153	0.059
150	2.4	0
200	3.2	0
300	4.8	0
600	9.6	0
1050	16.756	-0.260
1200	19.2	0
1800	28.8	0
2000	32.056	0.175
2400	38.4	0
4800	76.8	0

Nominal Baud Rate	**Actual 16x Clock (KHZ)**	**Error (percent)**
7200	115.2	0
9600	153.6	0
19200	307.2	0
38400	614.4	0

Bits 6,5,4 - Counter/Timer Mode and Clock Source Select

Table E-6 shows baud rate generator characteristics.

Table E-6: Counter/Timer Mode and Clock

Bits 6,5,4	**Mode**	**Clock Source**
0 0 0	Counter	DO NOT USE
0 0 1	Counter	TXCA - 1X clock of channel A transmitter
0 1 0	Counter	TXCA - 1X clock of channel B transmitter
0 1 1	Counter	3.6864 MHZ/16
1 0 0	Timer	DO NOT USE

Bits 6,5,4	**Mode**	**Clock Source**
1 0 1	Timer	DO NOT USE
1 1 0	Timer	3.6864 MHZ
1 1 1	Timer	3.6864 MHZ/16

Bits 3,2,1,0 - Change of State Interrupt Enable (0 = off, 1 = on)

This bit selects which bits of the Input Port Change Register (IPCR) cause the input change bit in the interrupt status register (ISR[7]) to be set. If a bit is in the "on" state, the setting of the corresponding bit in the IPCR will also result in the setting of ISR[7], which results in the generation of an interrupt output if IMR[7]=1. If a bit is in the "off" state, the setting of that bit in the IPCR has no effect on ISR[7].

E.3.9.10 INTERRUPT STATUS REGISTER (ISR) -

READ ONLY
DUART 0 address BASE + 112
DUART 1 address BASE + 154

7	6	5	4	3	2	1	0
input port chg	delta break B	RxRDY or FFULLB	TxRDY B	cnt rdy	delta break A	RxRDY or FFULLA	TxRDY A

Figure E-9: Interrupt Status Register

This register provides the status of all potential interrupt sources. The contents of this register are masked by the interrupt mask register (IMR). If a bit in the ISR is a '1' and the bit in the corresponding IMR is also a '1', the INTRN output will be asserted. If the corresponding bit in the IMR is a zero, the state of the bit in the ISR has no effect on the INTRN output. Note that the IMR does not mask the reading of the ISR -

the true status will be provided regardless of the IMR. All bits in the ISR are initialized to zero when the DUART is reset.

Bit 7 - Input Port Change Status

This bit is a '1' when a change of state has occurred at the IP0, IP1, IP2, or IP3 inputs and that event has been selected to cause an interrupt by the programming of ACR[3:0]. The bit is cleared when the CPU reads the IPCR.

Bit 6 - Channel B Change in Break

This bit, when set, indicates that the Channel B receiver has detected the beginning or the end of a received break. It is reset when the CPU issues a Channel B 'reset break change interrupt' command.

Bit 5 - Channel B Receiver Ready or FIFO Full

The function of this bit is programmed by MR1B[6]. If programmed as receiver ready, it indicates that a character has been received in Channel B and is waiting in the FIFO to be read by the CPU. It is set when the character is transferred from the receive shift register to the FIFO and reset when the CPU reads the RHR. If after this read there are more characters still in the FIFO the bit will be set again after the FIFO is 'popped.'

If programmed as FIFO full, it is set when a character is transferred from the receive holding register to the receive FIFO and the transfer causes the Channel B FIFO to become full; i.e., all three FIFO positions are occupied. It is reset when the CPU reads the RHR. If a character is waiting in the receive shift register because the FIFO is full, the bit will be set again when the waiting character is loaded into the FIFO.

Bit 4 - Channel B Transmitter Ready (0 = no, 1 = yes)

This bit is a duplicate of Transmitter Ready (SRB[2]).

Bit 3 - Counter Ready (0 = no, 1 = yes)

In the counter mode, this bit is set when the counter reaches terminal count and is reset when the counter is stopped by a stop counter command.

In the timer mode, this bit is set once each cycle of the generated square wave (every other time that the counter/timer reaches zero count). The bit is reset by a stop counter command. The command, however, does not stop the counter/timer.

Bit 2 - Channel A Change in Break
Same as bit 6 except for Channel A.

Bit 1 - Channel A Receiver Ready or FIFO Full
Same as bit 5 except for Channel A.

Bit 0 - Channel A Transmitter Ready
This bit is a duplicate of Transmitter Ready (SRA[2]).

E.3.9.11 INTERRUPT MASK REGISTER (IMR) -

WRITE ONLY
DUART 0 address BASE + 112
DUART 1 address BASE + 152

7	6	5	4	3	2	1	0
input port chg int	delta break B int	RxRDY or FULLB int	TxRDY int B	cnt rdy int	delta break A int	RxRDY or FULLA int	TxRDY int A

Figure E-10: Interrupt Mask Register

The programming of this register selects which bits in the ISR cause an interrupt output. If a bit in the ISR is a '1' and the corresponding bit in the IMR is also a '1', the INTRN output will be asserted. If the corresponding bit in the IMR is a zero, the state of the bit in the ISR has no effect on the INTRN output. Note that the IMR does not mask the programmable interrupt outputs OP3-OP7 or the reading of the ISR.

E.3.9.12 Counter/Timer Upper Byte Value (CTU) -

READ ONLY
DUART 0 address BASE + 114
DUART 1 address BASE + 154

In the counter mode, the current value of the upper and lower eight bits of the counter (CTU and CTL) may be read by the CPU. It is recommended that the counter be stopped when reading to prevent potential problems which may occur if a carry from the lower eight bits to the upper eight bits occurs between the times that both halves of the counter are read. However, note that a subsequent start-counter command will cause the counter to begin a new count cycle using the values in CTUR and CTLR.

E.3.9.13 Counter/Timer Upper Register (CTUR) -

WRITE ONLY
DUART 0 address BASE + 114
DUART 1 address BASE + 154

The CTUR and CTLR hold the eight MSBs and eight LSBs respectively of the value to be used by the counter/timer in either the counter or timer modes of operation. The minimum value which may be loaded into the CTUR/CTLR registers is 0002 (hex). Note that these registers are write-only and cannot be read by the CPU.

In the counter mode, the C/T counts down the number of pulses loaded into CTUR/CTLR by the CPU. Counting begins upon receipt of a start-counter command. Upon reaching terminal count (zero), the counter-ready interrupt bit (ISR[3]) is set. The counter continues counting past the terminal count until stopped by the CPU. ISR[3] is cleared when the counter is stopped by a stop counter command. The CPU may change the values of CTUR/CTLR at any time, but the new count becomes effective only on the next start-counter command. If new values have not been loaded, the previous count values are preserved and used for the next count cycle.

E.3.9.14 Counter/Timer Lower Byte Value (CTL) -

READ ONLY
DUART 0 address BASE + 116
DUART 1 address BASE + 156

See section E.3.9.12.

E.3.9.15 Counter/Timer Lower Register (CTLR) -

WRITE ONLY
DUART 0 address BASE + 116
DUART 1 address BASE + 156

See section E.3.9.13.

E.3.9.16 Channel B Mode Register 1 (MR1B) -

READ/WRITE
Channel 1 address BASE + 120
Channel 3 address BASE + 160

MR1B is accessed when the Channel B MR pointer points to MR1. The pointer is set to MR1 by RESET or by a "set pointer" command applied via CRB. After reading or writing MR1B, the pointer will point to MR2B.

The bit definitions for this register are identical to the bit definitions for MR1A, except that all control actions apply to the Channel B receiver and transmitter and the corresponding inputs and outputs.

E.3.9.17 Channel B Mode Register 2 (MR2B) -

READ/WRITE
Channel 1 address BASE + 120
Channel 3 address BASE + 160

MR2B is accessed when the Channel B MR pointer points to MR2, which occurs after any access to MR1B. Accesses to MR2B do not change the pointer.

```
    7-6            5          4          3-0
---------------------------------------------------
|              |   TX    |   TX    |                  |
| CHANNEL MODE | RTS CTL | CTS EN  | STOP BIT LENGTH  |
---------------------------------------------------
```

Figure E-11: Channel B Mode Register 2

Bits 7,6 - Channel B Mode Select
Same as Channel A mode select - See Section E.3.9.2.

Bit 5 - Channel B Transmitter Request-to-send control
This bit controls the deactivation of the RTS output by the transmitter.

In the 2/2 configuration, this bit should be cleared, disabling RTS for channels 1 and 3 from appearing at the outputs. This leaves the outputs for use as LL.

In the 4/0 configuration, this output is normally asserted by setting OPR[0] and negated by resetting OPR[0]. A "1" in this bit causes OPR[0] to be reset automatically one bit time after the characters in the Channel A transmit shift register and in the THR, if any, are completely transmitted, including the programmed number of stop bits, if the transmitter is not enabled. This feature can be used to automatically terminate the transmission of a message as follows:

1. Program auto-reset mode (bit 5 = 1).

2. Enable transmitter.
3. Assert RTS (OPR[0] = 1).
4. Send message.
5. Disable transmitter after the last character is loaded into the Channel B THR.
6. The last character will be transmitted and OPR[0] will be reset one bit time after the last stop bit, causing RTS to be negated.

Bit 4 - Channel B Clear-to-Send Control

This bit controls the deactivation of the CTS output (IPA1 or IPB1) by the transmitter.

In the 2/2 configuration, this bit should be cleared, disabling IPA1 or IPB1 from being used as an input for the channel 1 and 3 transmitters. IPA1 and IPB1 are used for the modem control inputs DSR.

In the 4/0 configuration, if this bit is zero, CTS has no effect on the transmitter. If this bit is a one, the transmitter checks the state of CTS (IPA1 or IPB1) each time it is ready to send a character. If IP1 is asserted (low), the character is transmitted. If it is negated (high), the Transmitter Data output remains in the marking state and the transmission is delayed until CTS goes low. Changes in CTS while a character is being transmitted do not affect the transmission of that character.

Bits 3,2,1,0 - Channel B Stop Bit Length Select

Same as Channel A - See Section E.3.9.2.

E.3.9.18 Status Register B (SRB) -

READ ONLY
Channel 1 address BASE + 122
Channel 3 address BASE + 162

Definitions same as for Channel A. See Section E.3.9.3.

E.3.9.19 Channel B Clock Select Register (CSRB) -

WRITE ONLY
Channel 1 address BASE + 122

Channel 3 address BASE + 162

Definitions same as for Channel A. See Section E.3.9.4.

E.3.9.20 Channel B Command Register (CRB) -

WRITE ONLY
Channel 1 address BASE + 124
Channel 3 address BASE + 164

Definitions same as for Channel A. See E.3.9.5.

E.3.9.21 Channel B Receive Holding Register (RHRB) -

READ ONLY
Channel 1 address BASE + 126
Channel 3 address BASE + 166

Definitions same as for Channel A. See Section E.3.9.6.

E.3.9.22 CHANNEL B TRANSMITTER HOLDING REGISTER (THRB) -

WRITE ONLY
Channel 1 address BASE + 126
Channel 3 address BASE + 166

Definitions same as for Channel A. See Section E.3.9.7.

E.3.9.23 INPUT PORT (IP) -

READ ONLY
Duart 0 address BASE + 132
Duart 1 address BASE + 172

The inputs to this unlatched 7-bit port can be read by performing a read operation of the above addresses. A high input results in a logic 1 while a low input results in a logic 0. Bit 7 will always be read as a logic 1.

Four change-of-state detectors are provided which are associated with IP3, IP2, IP1, IP0. A high-to-low or a low-to-high transition of these inputs lasting longer than 25-50 useconds will set the corresponding bit in the IPCR. The bits are cleared when the register is read. Any change of state can also be programmed to generate an interrupt.

For input port bit definitions, see the appropriate modem configuration sections.

E.3.9.24 Output Port Configuration Register (OPCR) -

WRITE ONLY
Duart 0 address BASE + 132
Duart 1 address BASE + 172

```
   7      6     5     4    3,2   1,0
-----------------------------------------
| OP7 | OP6 | OP5 | OP4 |   OP3   |  OP2   |
| CTL | CTL | CTL | CTL |   CTL   |  CTL   |
-----------------------------------------
```

Figure E-12: Output Port Configuration Register

Bit 7 - reserved - this functionality not used.

Bit 6 - reserved - this functionality not used.

Bit 5 - reserved - this functionality not used.

Bit 4 - OPA4, OPB4 control

For 2/2 configuration, bit 4 = 0. This causes the complement of OPR[4] to be ouput to OP4.

For the 4/0 configuration, this bit is reserved, this functionality not used.

Bits 3,2 - OPA3, OPB3 control

For both configurations, bit[3,2] = 00. This causes the complement of OPR[3] to be output to OP3.

Bits 1,0 - OPA2, OPB2 control

For both configurations, bit[1,0] = 00. This causes the complement of OPR[2] to be output to OP2.

E.3.9.25 START COUNTER Command -

DUART 0 address BASE + 134
DUART 1 address BASE + 174

A read of this address starts the counter.

E.3.9.26 SET OUTPUT PORT BITS Command -

WRITE ONLY
DUART 0 address BASE + 134
DUART 1 address BASE + 174

A bit is set in the output port by performing a write operation with the accompanying data specifying the bits to be set (1 = set, 0 = no change).

E.3.9.27 STOP COUNTER Command -

READ ONLY
DUART 0 address BASE + 136
DUART 1 address BASE + 176

A read of this address stops the counter.

E.3.9.28 Reset Output Port Bits Command -

WRITE ONLY
DUART 0 address BASE + 136
DUART 1 address BASE + 176

A bit is reset in the output port by performing a write operation with the accompanying data specifying the bits to be reset (1 = reset, 0 = no change).

E.4 MODEM CONTROLS FOR 2/2 CONFIGURATION

This section describes the modem control signals when the module is configured to have two ports with full asynchronous modem control and two ports with no modem control (data leads only).

E.4.1 Input Port Assignments

The input port bits are assigned as follows :

Table E-7: 2/2 Input Port Bits

	DUART 0	**DUART 1**
BIT 0	channel 0 CTS	channel 2 CTS
BIT 1	channel 0 DSR	channel 2 DSR
BIT 2	channel 0 CD	channel 2 CD
BIT 3	channel 0 RI	channel 2 RI
BIT 4	channel 0 TI	channel 2 TI
BIT 5	channel 0 SPDMI	channel 2 SPDMI
BIT 6	not used	not used

E.4.2 Output Port Bit Definitions

Table E-8: 2/2 Output Port Bits

	DUART 0	**DUART 1**
BIT 0	channel 0 RTS	channel 2 RTS
BIT 1	channel 0 LL	channel 2 LL
BIT 2	channel 0 DTR	channel 2 DTR
BIT 3	channel 0 RL	channel 2 RL

MODEM CONTROLS FOR 2/2 CONFIGURATION

	DUART 0	**DUART 1**
BIT 4	channel 0 DSRS	channel 2 DSRS
BIT 5	not used	not used
BIT 6	not used	not used
BIT 7	not used	not used

E.4.3 Input Port Set-Up

E.4.3.1 CTS -

Channels 0 and 2 MR2A, bit 4

If this bit is a zero, CTS has no effect on the channel transmitter. If this bit is a 1, the transmitter checks the state of CTS (IPA0 or IPB0) each time it is ready to send a character.

Channels 1 and 3 MR2B, bit 4

This bit should be set to zero so that IPA1 has no effect on the channel 1 transmitter, and IPB1 has no effect on the channel 3 transmitter.

E.4.3.2 DSR - Set-up is as required above. Since IPA1 is not set up to affect channel 1 transmitter, it can be used for the channel 0 DSR. Also IPB1 can be used for the channel 2 DSR.

E.4.3.3 CD - The counter/timer clock source should NOT be programmed to be IPA2 (or IPB2). This leaves these bits free for use as channel 0 TI and channel 2 TI.

E.4.3.4 RI - The channel 0 or channel 2 transmitter clock source should *not* be programmed to be IPA2 (or IPB2). This leaves these bits free for use as channel 0 SPDMI and channel 2 SPDMI.

E.4.4 Set-Up for Outputs

E.4.4.1 RTS - Bit 5 for MR2A (channels 0 and 2) can be set or cleared. See MR2A bit 5 definition.

E.4.4.2 LL - Bit 5 for MR2B (channels 1 and 3) should be cleared (0).

E.4.4.3 DTR - Bits 1 and 0 should be cleared (00) for OPCR in both DUARTs.

E.4.4.4 RL - Bits 3 and 2 should be cleared (00) for OPCR in both DUARTs.

E.4.4.5 DSRS - Bit 4 should be cleared (0) for OPCR in both DUARTs.

E.5 MODEM CONTROLS FOR 4/0 CONFIGURATION

This section describes the modem control signals when the module is configured to have four ports with partial asynchronous modem control.

E.5.1 Input Port Assignments

The input port bits are assigned as follows :

Table E-9: 4/0 Input Port Bits

	DUART 0	**DUART 1**
BIT 0	channel 0 CTS	channel 2 CTS

MODEM CONTROLS FOR 4/0 CONFIGURATION

	DUART 0	**DUART 1**
BIT 1	channel 1 CTS	channel 3 CTS
BIT 2	channel 0 CD	channel 2 CD
BIT 3	channel 1 CD	channel 3 CD
BIT 4	channel 0 DSR	channel 2 DSR
BIT 5	channel 1 DSR	channel 3 DSR
BIT 6	not used	not used

E.5.2 Output Port Bit Definitions

Table E-10: 4/0 Output Port Bits

	DUART 0	**DUART 1**
BIT 0	channel 0 RTS	channel 2 RTS
BIT 1	channel 1 RTS	channel 3 RTS
BIT 2	channel 0 DTR	channel 2 DTR
BIT 3	channel 1 DTR	channel 3 DTR
BIT 4	not used	not used
BIT 5	not used	not used

MODEM CONTROLS FOR 4/0 CONFIGURATION

	DUART 0	**DUART 1**
BIT 6	not used	not used
BIT 7	not used	not used

E.5.3 Input Port Set-Up

E.5.3.1 CTS - For all channels (DUART 1 MR2A and MR2B, and DUART 2 MR2A and MR2B), MR2 bit 4 controls CTS. If this bit is a zero, CTS has no effect on the channel transmitters. If this bit is a 1, the transmitter checks the state of CTS each time it is ready to send a character.

E.5.3.2 CD - The counter/timer clock source should NOT be programmed to be IPA2 (or IPB2). The channel 0 or channel 2 transmitter clock source should NOT be programmed to be IPA3 (or IPB3). This leaves these bits free for use as channel 0-3 CD.

E.5.3.3 DSR - The channel 0 or channel 2 receiver clock source should NOT be programmed to be IPA4 (or IPB4). The channel 1 or channel 3 transmitter clock source should NOT be programmed to be IPA5 (or IPB5). This leaves these bits free for use as channel 0-3 DSR.

E.5.4 Set-Up for Outputs

E.5.4.1 RTS - Bit 5 for MR2 channels 0-3 can be set or cleared. See MR2A and MR2B bit 5 definitions.

E.5.4.2 DTR - Bits 3-0 should be cleared (0000) for OPCR in both DUARTs.

E.5.5 Connector Box

The connector box gives you the option of having full or partial modem control. The way that this is accomplished is by having two 40-pin connectors in the connector box.

If you plug the 40-pin cable into the end of the box on the 12-pin side of the D-subs, then you will be plugged into connector J1, in the full modem control configuration.

If you plug the 40-pin cable into the end of the box on the 13-pin side of the D-subs, then you will be plugged into connector J6, in the partial modem control configuration.

INDEX

Guide to Writing a P/OS I/O Driver and Advanced Programmer's Notes
AD-BT73B-T1

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

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

Did you find errors in this manual? If so, specify the error and the page number.

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

- ☐ Assembly-language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) ____

Name ____ Date ____
Organization ____
Street ____
City ____ State ____ Zip Code ____
or
Country

Please cut along this line.

— — Do Not Tear — Fold Here and Tape — — — — — — — — — — — — — — — — —

digital

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
CORPORATE USER PUBLICATIONS
MLO5–5/E45
146 MAIN STREET
MAYNARD, MA 01754–2571

— — Do Not Tear — Fold Here —

Printed in U.S.A.