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

Order No. AA-H267B-TC

RSX-11M-PLUS Version 2.0

digital equipment corporation · maynard, massachusetts

First Printing, October 1979
Revised, March 1982

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

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

Printed in U.S.A.

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

ZK2150

HOW TO ORDER ADDITIONAL DOCUMENTATION

n Continental USA and Puerto Rico call 800-258-1710

n New Hampshire, Alaska, and Hawaii call 603-884-6660

n Canada call 613-234-7726 (Ottawa-Hull)
800-267-6146 (all other Canadian)

DIRECT MAIL ORDERS (USA & PUERTO RICO)*

Digital Equipment Corporation
P.O. Box CS2008
Nashua, New Hampshire 03061

*Any prepaid order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575)

DIRECT MAIL ORDERS (CANADA)

Digital Equipment of Canada Ltd.
940 Belfast Road
Ottawa, Ontario K1G 4C2
Attn: A&SG Business Manager

DIRECT MAIL ORDERS (INTERNATIONAL)

Digital Equipment Corporation
A&SG Business Manager
c/o Digital's local subsidiary or
approved distributor

nternal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Northboro, Massachusetts 01532

CONTENTS

Page

Page

FIGURES

TABLES

PREFACE

MANUAL OBJECTIVES

The primary goal of this manual is to introduce RSX-11M-PLUS 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 RSX-11M-PLUS system
- Detect typical errors that cause the system to crash
- Use Executive service routines that an I/O service routine typically employs

A secondary objective is to introduce advanced hardware and software features and sophisticated Executive facilities, and to describe both the conventional and advanced features of I/O data structures and mechanisms. Knowledge of advanced features should facilitate the understanding of conventional I/O processing and eliminate some of the confusion inherent in seeing data structures without knowing their usage.

The manual does not describe how to write software that incorporates advanced driver features. The only complete package of such information is DIGITAL-supplied software, such as DVINT.MAC and DBDRV.MAC (for overlapped seek, dual access, and common interrupt handling); IOSUB.MAC and TTDRV.MAC (for full duplex I/O); and MMDRV.MAC (for subcontroller device operation). The manual also does not describe how to attach hardware to the PDP-11, how to perform diagnostic functions to uncover hardware faults, nor how to incorporate DIGITAL-standard error-reporting functions in user-written software.

INTENDED AUDIENCE

This manual is written for the senior-level system programmer who is familiar with the hardware characteristics of both the PDP-11 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 an RSX-11M-PLUS 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 real-time system software, a senior-level system programmer should be able to appreciate how user-written software interfacing with the Executive can affect overall system performance.

STRUCTURE OF THIS DOCUMENT

This document is structured to be self-contained so that you need not refer to any other manual to build and incorporate a user-written driver into your system. The manual has three types of information: conceptual, procedural, and reference. The following are abstracts of the chapters in the document:

- Chapter 1, "RSX-11M-PLUS I/O Drivers," introduces terms and concepts fundamental to understanding physical I/O in RSX-11M-PLUS, and describes the protocol that a driver must follow to preserve system integrity. It summarizes advanced driver features and RSX-11M-PLUS 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 RSX-11M-PLUS," 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. Also included are a summary of the system generation dialogue concerning including user-supplied drivers and a description of the loading mechanism and the diagnostic operations performed during loading.

- 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 RSX-11M-PLUS 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.
- 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, "Converting a User-Supplied RSX-11M Driver," describes the modifications that you must make to enable an RSX-11M user-supplied driver to run on an RSX-11M-PLUS system.

ASSOCIATED DOCUMENTS

Accompanying your RSX-11M-PLUS system are documents that describe both the software and hardware on the system. The software documents are listed and described in the RSX-11M-PLUS Information Directory and Index. Consult the directory for concise summaries of software-related publications. Processor and peripherals handbooks summarize hardware information published in various maintenance, installation, and operator manuals that are provided with your system.

SUMMARY OF TECHNICAL CHANGES

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

1. Two new arguments (BUF and OPT) have been added to the DDT$ macro call in Chapter 4.

2. The I/O Function Masks for Mass Storage, Magtape, and Unit Record Devices have been added in Chapter 4.

3. Additions have been made to the UCB in Chapter 4 as follows:

   - A new symbolic name U.MUP has been added (redefinition of U.CLI)

   - The DV.MXD offset to U.CW1 has been renamed to DV.MSD for mass storage.

   - U.UCBX has been added for mass storage errorlogging devices.

4. New status control block extension bit definitions (S2.OPT, S2.OP1, S2.OP2, and S3.OPT) have been added to the SCB in Chapter 4.

5. New controller bit definitions (KS.MOF, KS.EXT and KS.SLO) have been added to the KRB in Chapter 4.

6. The Queue Optimization entry point, Deallocation entry point and the Next command entry point have been added to Chapter 4.

7. New tracing faults of $HEADR have been added to Chapter 6.

8. Some Executive routines listed in Chapter 7 have been moved to new Executive modules and 12 have been added. The following is a list of the affected modules and subroutines, plus the additions:

Routine	Old Module	New Module
$ACHKB	IOSUB	EXSUB
$ACHCK	IOSUB	EXSUB
$ASUMR	IOSUB	MEMAP
$BLKCK	IOSUB	MDSUB
$BLKC1	(new)	MDSUB
$BLKC2	(new)	MDSUB
$BLXIO	(new)	BFCTL
$CKBFI	(new)	EXESB
$CKBFR	(new)	EXESB
$CKBFW	(new)	EXESB
$CKBFB	(new)	EXESB
$CVLBN	IOSUB	MDSUB
$DEUMR	IOSUB	MEMAP

SUMMARY OF TECHNICAL CHANGES

Routine	Old Module	New Module
$INIBF	(new)	IOSUB
$MPUBM	IOSUB	MEMAP
$MPUB1	IOSUB	MEMAP
$RELOC	IOSUB	MEMAP
$RELOP	IOSUB	MEMAP
$REQUE	(new)	IOSUB
$REQU1	(new)	IOSUB
$STMAP	IOSUB	MEMAP
$STMP1	IOSUB	MEMAP
$TSPAR	(new)	REQSB
$TSTBF	(new)	IOSUB

CHAPTER 1

RSX-11M-PLUS I/O DRIVERS

Device drivers on RSX-11M-PLUS are the primary method of interfacing Executive software with hardware attached to the computer. Most DIGITAL-supplied hardware[1] is supported by drivers accompanying the remaining software that the user receives with the system. This chapter introduces the concept of device drivers and explains driver operations and features.

1.1 VECTORS AND CONTROL AND STATUS REGISTERS

A device controller has a unique address on the PDP-11 UNIBUS that identifies itself and distinguishes it from other hardware attached to the computer. At this unique address is usually a control and status register (CSR) containing data elements that allow software to operate and interrogate the related device. The CSR resides in physical address space that is reserved for device registers and is referred to as the I/O, or peripheral, page. Other registers associated with the device are placed in contiguous addresses lower and/or higher than the CSR address. Software usually controls a device by accessing the CSR to enable interrupts, initiate a function, and respond to the resulting interrupt to continue or finish the function.

Associated with many devices can be one or more 2-word areas 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, it sends the processor the address of the interrupt vector. 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. On a multiprocessor system, each central processor unit has its own vector space.

[1] The CINT$ directive enables a privileged task to gain control when a device interrupts and thereby to access device registers. The K-series Laboratory modules use this feature to perform I/O. The CINT$ directive is a secondary but equally viable method of interfacing software to hardware.

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. A driver can reside with the Executive itself or can be separated from it. The former driver is resident and the latter is loadable.

The distinction between resident and loadable drivers is mainly one of flexibility. A resident driver is built in during system generation as a permanent part of the Executive.[1] It resides in the Executive address space and cannot be removed. A resident driver responds to interrupts slightly faster than a loadable driver. Although linked into the Executive structures, a loadable driver resides in memory outside the virtual address space of the Executive. A user can add or remove a loadable driver by means of an MCR or VMR command. 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. On a system without Executive data space support, making a driver loadable frees virtual space in the Executive which can be used for additional pool.

1.2.1 Executive and Driver Layout

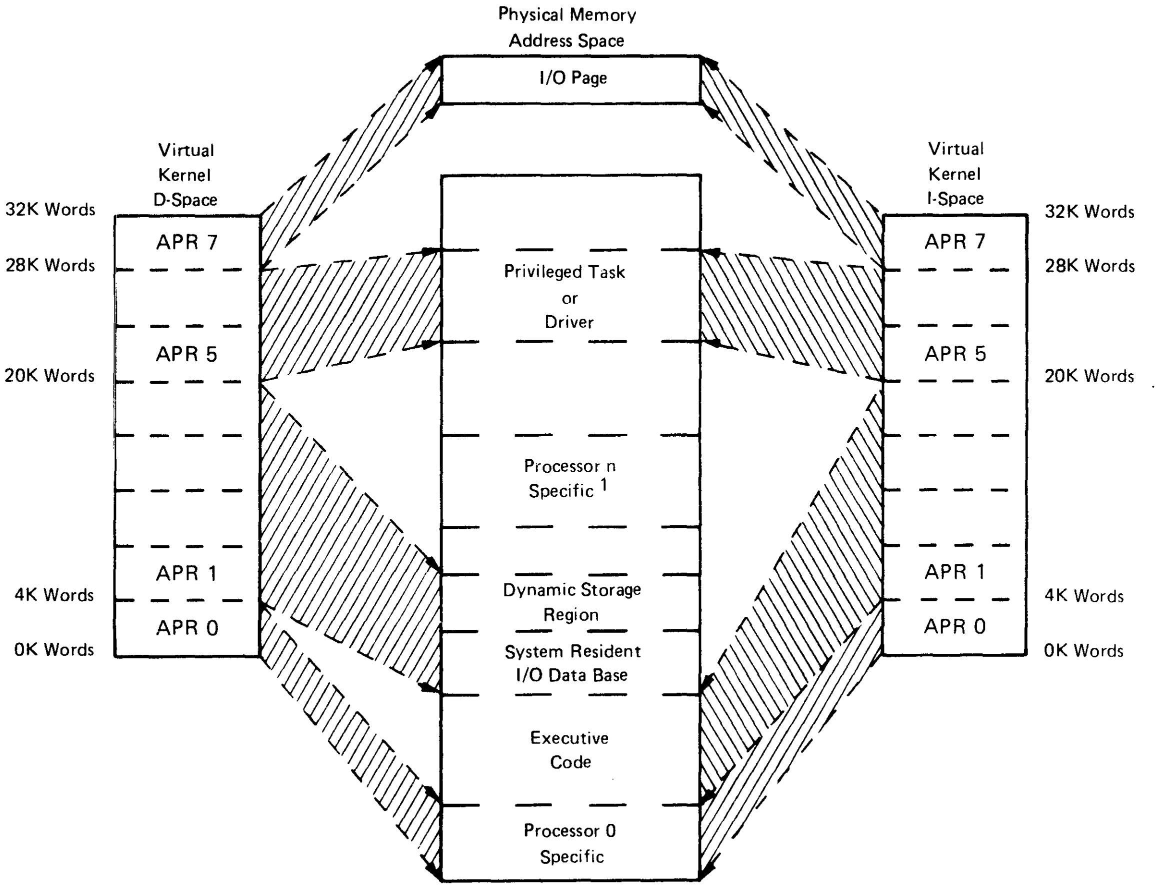
A device driver is a logical extension of the Executive that need not be contiguous in physical memory with the Executive code. Active Page Registers (APRs) 0 through 4 map the Executive, whereas APR 7 is reserved to map the I/O page.[2] Resident drivers are mapped within the Executive space. Loadable drivers reside in a separate partition of memory and are mapped by APR 5. Therefore, a loadable 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 on a system with Kernel data space support is shown in Figure 1-1.

Virtual addresses 0 through 4K words (APR 0) of I and D space overmap the same physical memory. The mapped area contains the interrupt vectors, processor stack, processor-specific memory locations, and interrupt control block (ICB) pool space as well as some Executive code. I-space virtual addresses 4K through 20K words (APR1 through APR 4) map the remaining Executive code, which is therefore limited to 16K words. D-space virtual addresses 4K through 20K words (APR 1 through APR 4) map the dynamic storage region (or pool) and system data structures to a maximum of 16K words.

[1] On systems with Executive data space support, all drivers must be loadable.

[2] Active Page Register is a term referring to the KT11 Memory Management register pair (Page Address Register (PAR) and Page Descriptor Register (PDR).) Refer to the relevant processor handbook for information on hardware mapping and memory management. Refer to the RSX-11M/RSX-11M-PLUS Task Builder Manual for a description of mapping and APR assignments by software.



[1] On multiple processor systems, each additional processor requires its own processor-specific area in the CPU partition.

ZK-245-81

Figure 1-1 Virtual to Physical Mapping for the Executive

Virtual addresses 20K through 28K words (APR 5 and APR 6) of I and D space overmap the same physical memory, which is reserved to map loadable 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) of I and D space overmap 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. It also has available all the Executive service routines to help it process I/O requests.

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 code of a driver must be in one continuous portion of main memory. Because the Executive is designed to respond to real-time activities, the driver code must run as fast as possible. Therefore, it cannot be overlaid.

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 1.4.7 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 instructions, 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, posts I/O status, and requests another I/O packet. This sequence of execution steps continues until the I/O queue is empty. The driver process then 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 is in the low-order four bits.)

Because an interrupt must be serviced in Kernel address space, how the interrupt is handled depends on whether the driver is resident or loadable. A resident driver, being mapped with the Executive in Kernel address space, handles the interrupt directly (that is, the entry point address of the driver is the PC word of the interrupt vector). For a resident driver, then, the hardware dispatches directly to the interrupt service routine in the driver. Figure 1-2 shows this mechanism.

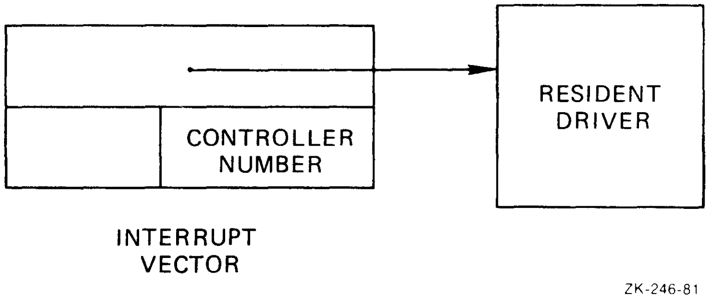


Figure 1-2 Interrupt Dispatching for a Resident Driver

When the interrupt service routine in the resident driver gains control, it runs at priority 7, which locks out further interrupts. The driver is therefore uninterruptable and, because the system must respond to real-time events, processing at this level cannot take too long.[1]

To ensure that a driver does not lock out other interrupts on the system or destroy the context of any interrupted process, a protocol has been established. By system convention, no process should run at an uninterruptable level for more than 100 microseconds. A common Executive coroutine, called interrupt save ($INTSV), exists to lower the priority level of the driver process to that of the interrupting device and to save two registers of the interrupted process. Therefore, by system convention, all resident drivers call the $INTSV coroutine, which saves the PS and extracts the controller number. Because most instructions change the PS bits that encode the controller number, under most circumstances the driver can do very little else without saving the controller number.

The $INTSV 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.) 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. This protocol makes the driver partially interruptable (that is, interruptable by devices with a higher priority) and preserves the context of the interrupted process.

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.[2]

For a loadable 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 1-3 shows this mechanism.

The ICB actually contains a JSR instruction to an Executive interrupt save routine ($INTSI) and some data cells that enable the routine to do the following:

- Save R4 and R5

- Save the Kernel mapping (APR 5)

[1] On a multiprocessor system, a driver running at priority 7 is interruptable by a device of the same type on another CPU. To handle this situation, the driver being interrupted does not have to do any special processing beyond what is described in this manual.

[2] An Executive interrupt exit routine, $INTXT, exists to standardize the way a driver exits from an interrupt. However, on RSX-11M-PLUS systems this routine is simply a RETURN instruction.

- Load APR 5 to map the driver
- Transfer control to the driver
- Restore the mapping after return from the driver
- Restore R4 and R5

Thus, the interrupt vector for a controller serviced by a loadable driver points to an ICB rather than to the driver. Accordingly, the loadable driver does not (and must not) call the $INTSV routine as the resident driver does because the $INTSI routine saves the context on behalf of the loadable driver. When it gains control, the loadable driver is also partially interruptable as if it had called the $INTSV routine. After it gains control, the loadable driver is exactly like the resident driver. (That is, it must also observe the protocols established on the system.)

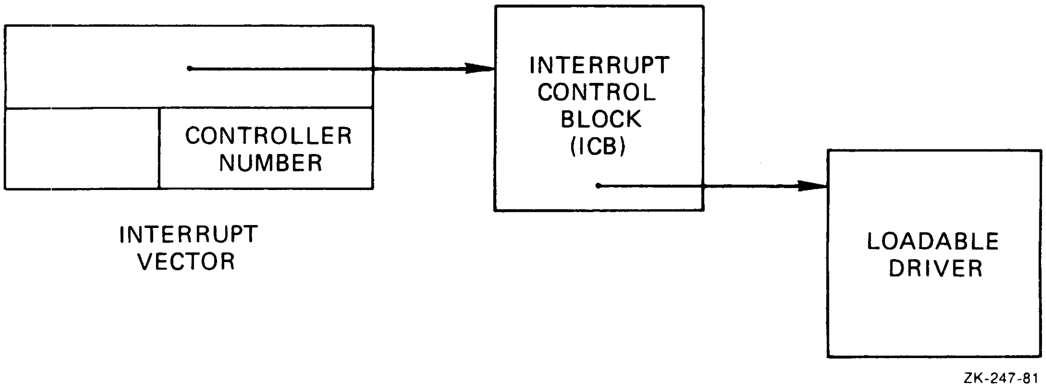


Figure 1-3 Interrupt Dispatching for a Loadable Driver

The ICB 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 real controller number, the Executive interrupt service routine adds the controller group number in the ICB and the controller number in the PS. (Note that, because a resident driver does not use the ICB mechanism, there can be at most 16 controllers of one type if the driver is resident. Furthermore, only the LOAD command in VMR supports more than 16 controllers of one type.)

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. When an interrupt occurs, the driver can determine the number of the saved controller from information encoded in the low-order four bits of the PS. The interrupt service routine in the driver uses the number to index a table in the CTB and to access the proper unit data and context.

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 (whether resident or loadable) 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 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.[1] A driver must therefore call the fork routine before it calls any other Executive service routine (except for $INTSV), 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 will eventually cause 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 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 resides 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.

[1] 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.

The status of a fork process lies between an interrupting routine and a task requesting system resources. 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 registers, a fork process can use all registers. The fork processes are completely interruptable. Tasks run only when the fork list is empty.[1]

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 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 fork level.

Between the time that a driver process calls $FORK and the Executive starts the process at fork level, 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 fork level. If the driver does not ensure against unexpected interrupts, it may double fork as described above. As a result of the double fork, the driver may either miss an interrupt from the device or miss interrupts from several devices. As a further consequence, code after the call to $FORK is executed twice for the same context with generally catastrophic results. For example, calling $IODON twice for the same I/O packet eventually causes the system to crash.

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

[1] On a multiprocessor system, the fork list is not necessarily empty when the Executive returns control to a task. The Executive processes only those fork blocks that are to run on the current processor. To ensure that fork blocks remaining in the list are readily processed, the Executive running on one processor interrupts (using the interprocessor interrupt hardware) any other processor that has fork blocks waiting for processing.

unexpected interrupts from these devices. If error logging is active, any unexpected interrupts are recorded as undefined interrupt errors. This feature helps in detecting faulty hardware.

1.4 ADVANCED DRIVER FEATURES

Advanced drivers have certain optional and built-in special features. This section introduces these features so that you can better understand the structures 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. This feature allows any or all drives attached to the same controller 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 in its attention summary register. 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-specific interrupt service routine built into the Executive examines the device registers to determine whether the interrupt was initiated by the controller or the drive unit. Using the controller number retrieved from the PS in the interrupt vector, the routine forms an index (called the controller index) to use 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 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 Attention Summary Register. 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 Dual-Access Support

Some devices have multiple-access paths for both control and data transfer functions. Such devices are called dual access. A dual-access unit is connected to two controllers at one time and may be accessed from either controller at the option of the system software. Since a single device unit may have only one physical unit number, a dual-access unit must have the same unit number for both controllers. A dual-access unit may be accessed only from one port at a time. The system supports dual-access operation for those devices equipped with the necessary hardware capability. This feature is most useful on a multiprocessor system where each access path is to a different central processor unit.

To support dual-access operations, the I/O data structures must reflect the existence of alternate controllers. Particularly, the driver process context for I/O on a unit can be associated with either of two controllers. To decide which controller will provide access to the drive unit, the driver must call an Executive routine to request access to a particular controller. When the Executive grants access, the driver process context for a unit is associated with the assigned controller. A driver must have access to the assigned controller before actually changing the registers in the I/O page.

When a driver and a unit are given access to a controller, the controller status is set to busy. The unit becomes the device owned by the controller for the operation. A controller without an owned unit is considered a free controller. By this ownership mechanism, controller interrupts are sent to the correct unit for processing. After the operation completes, the driver requests the Executive to release the controller and thus frees it.

1.4.3 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.

1.4.4 Controller Reassignment and Load Sharing

Controller assignment for dual-access devices is dynamic. If one port (access path) to a device is busy, the system can request access on the other port. This switching between ports allows the system to share the load between the two controllers.

NOTE

> A dual-access device has both ports attached to the same system. DIGITAL does not support systems loosely coupled through a peripheral.

The system also maintains an I/O count for each controller to determine how busy it is. If one controller is not as busy as the other, the system can queue the access requests to the less busy controller. Whenever load sharing is done on a dual-access unit, the Executive makes any reassignment necessary before actually requesting access to the controller.

1.4.5 Common Interrupt Dispatching

To handle interrupts from a controller that supports more than one type of device, the Executive uses a mechanism called common interrupt dispatching. The RH70 MASSBUS controller can have different types of devices (RP04, RP05, and RP06 moving head disks; RM02, RM03, RM05, moving head disks; RM80 and RP07 fixed media disks; ML11 non-rotating memory; RS03 and RS04 fixed head disks; and TE16, TU45, and TU77 magnetic tape drives) connected to the same type of controller. Interrupt dispatching for such devices is more difficult than for standard interrupt devices because associated with one set of interrupt vectors are multiple drivers. To dispatch interrupts, therefore, a routine in the Executive must intervene. Figure 1-4 shows an example of common interrupt dispatching.

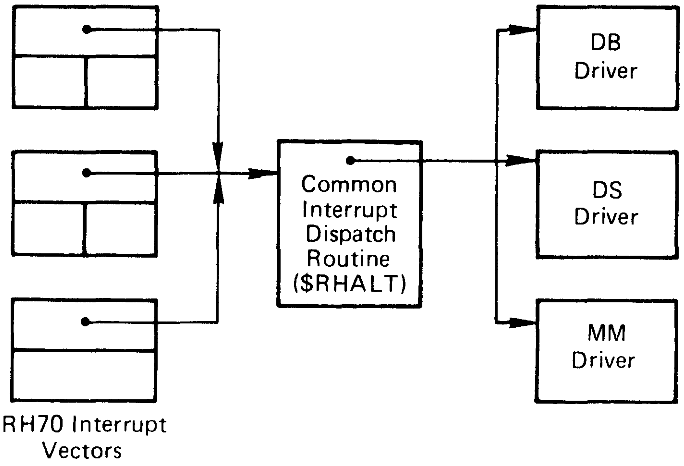


Figure 1-4 Interrupt Dispatching for Common Interrupt Devices

The vectors for such controllers point to a common interrupt dispatching routine in the Executive module DVINT. This common routine avoids having to duplicate code in drivers. This routine, in essence acting like an RH70 controller driver or a sophisticated ICB, determines which driver will receive control upon an interrupt. Operating like the routine that handles interrupts for overlapped seek devices, this routine determines the type of device that interrupted and dispatches to the proper driver.

1.4.6 Subcontroller Devices

Certain devices have 2-level controllers, such as magtapes, where a TM03 connects to an RH70 MASSBUS controller and also connects to TE16 magtape drives. In such an arrangement, the TM03 is a subcontroller, or master unit, that controls slave units; a register in the master unit reports the number of the slave unit that generates an interrupt.

A subcontroller is associated with a data structure called a subcontroller request block (KRB1) that serializes access to the subcontroller. Therefore, a driver must request and receive access to both the subcontroller and the controller for a unit before executing any operations. The KRB1 is a subset of the KRB and every unit on the subcontroller points to the KRB1 of the subcontroller to which it is attached.

1.4.7 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.

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 when the device is configured on-line.

A driver that handles full duplex operations provides a specific example of software that handles concurrent I/O for individual units. Some devices, such as the DIGITAL-supplied LPA11-K microprocessor-based laboratory subsystem, can handle a number of simultaneously active I/O requests. The software to handle such concurrent I/O may require augmented driver data base structures so that the context of each I/O process remains distinct and controllable. The driver for the LPA11-K relies on an extended user control block (UCB) to preserve the context of a maximum of eight simultaneously active I/O processes. User-written software for such a device must properly synchronize fork processing to prevent substituting the I/O context of one process for that of another. Moreover, the $GSPKT routine also might be used as described above to make a unit appear idle when it is busy.

1.4.8 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 checkpointable, buffered I/O can be performed. $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 <u>RSX-11M/M-PLUS Executive Reference Manual</u>.

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 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.

1.4.9 I/O Queue Optimization

Without I/O queue optimization, the operating system groups input and output requests in the queue by highest priority on a first-in, first-out basis. The first request at the highest priority appears first in the queue and is processed first. Other requests within that priority are then processed sequentially until the last request at that priority is serviced.

With I/O queue optimization, however, the next I/O request at the highest priority is not necessarily the next sequential request to be processed. I/O queue optimization allows the queue to be scanned, and each request to be examined. The I/O request, according to the method of optimization then in effect, is the next one dequeued and passed to the I/O driver for processing. The highest priority requests are still serviced first; however, throughput is improved by the reordering of requests within a priority.

There are three methods of I/O queue optimization available:

- Nearest Cylinder
- Elevator
- Cylinder Scan

The Nearest Cylinder method processes the I/O request that is closest to the one at which the disk head is currently positioned. The Elevator method processes requests as the disk head moves from the perimeter to the innermost track of the disk. Once the disk head reaches the innermost track, the direction is reversed and requests are processed along the disk as the head moves back to the perimeter. The Cylinder Scan method operates similar to the Elevator method, except requests are only processed as the disk head moves from the perimeter to the innermost track. Once at the innermost track, the disk head returns to the perimeter and begins processing new requests.

The method you choose for your system is dependent upon the I/O processing requirement of your application, the frequency with which tasks access certain data areas on the disk, and the physical location of data on the disk. Refer to the RSX-11M/M-PLUS System Management Guide for information on selecting I/O queue optimization methods.

Before an I/O request can be queued to the driver, all three queue optimization methods require the starting cylinder number of the I/O request. To find the cylinder number, the logical block number (LBN) of each I/O request is converted to cylinder, track, and sector form. The routine $DRQRQ in the Executive module DRSUB begins this conversion. Because the cylinder, track, and sector form is specific to the device geometry, this conversion must be completed by a separate routine in the driver. The routine $DRQRQ locates the conversion routine in the driver through offset D.VCHK in the driver dispatch table.

The routine $DRQRQ calls the conversion routine for all I/O requests. However, if the functions are not logical transfer functions, such as ACP functions or Attach and Detach operations, the conversion routine does not complete the conversion, but rather returns to $DRQRQ.

Drivers without queue optimization call the routine $BLKCK in the Executive module MDSUB to check the limits of the I/O request. If $BLKCK locates an error, the routine $IOALT in the Executive module IOSUB is called for the I/O request and the driver is returned to the initiation entry point. If you chose queue optimization, a return to the initiation entry point is not desirable because the necessary functions of $DRQRQ will not be completed. Therefore, your completion routine must call the routine $BLKC2 in the Executive module MDSUB instead of $BLKCK to ensure the correct return to $DRQRQ if an error is detected.

The routine $GTPKT in the Executive module IOSUB performs the actual optimization. The driver calls the Executive routine $GTPKT for an I/O request to process. $GTPKT scans the queue of I/O packets to select those of the highest priority. The routine then chooses the correct packet within that priority based on the optimization method currently in effect, dequeues that packet, and returns control to the driver to process that I/O request.

1.5 DISTRIBUTED I/O

On a multiprocessor system, a task may issue an I/O request to any device on any processor. The Executive must be responsible for distributing the I/O request to the correct processor. To ascertain to which processor a device is attached and to have the driver execute on the correct processor, the Executive must perform some processor-specific functions. The following sections introduce the data structure and the processing routines used by the Executive for processor-specific functions.

1.5.1 UNIBUS Run Mask

To help describe devices attached to a processor, the software relies on a concept called UNIBUS run. A UNIBUS run consists of a group of distinct devices, all of which are electrically connected to the same UNIBUS and are not separated by any bus reconfiguration devices. Each UNIBUS run is attached to the same processor at the same time because of the way the devices are physically attached to the UNIBUS. (Devices attached to a MASSBUS of a processor are also on the processor's UNIBUS run.) The UNIBUS run, then, is the smallest fragment of a particular UNIBUS capable of being switched (or not switched) between processors.

Essential to understanding UNIBUS runs is the concept of a switched bus. A switched bus is a portion of a UNIBUS that can be physically connected to one of multiple UNIBUSes. A device on the UNIBUS, called the DT07 UNIBUS switch, controls the connection and allows a switched bus to be connected to any one of a maximum of four UNIBUSes. Any UNIBUS device or devices except a processor or another bus switch may be connected to a switched bus. Moreover, because of the electrical delay associated with the bus switch, some high-speed devices (such as the DMC-11) cannot be on a switched bus.

In a multiprocessor system, the DT07 allows the switched bus to be physically switched from the UNIBUS of one processor to the UNIBUS of another processor. When the switch is connected to a particular processor's UNIBUS, all peripherals on the switched bus operate as if they were permanently connected to that UNIBUS. By means of reconfiguration software, a switched bus can be disconnected from one UNIBUS and be available for connection to another processor's UNIBUS. Because a user task can direct an I/O request to any device on the system, the Executive must be able to perform the operation on the specific processor to which the device is connected.

A UNIBUS run is represented in a cell called a UNIBUS run mask (or URM). The URM is a 16-bit word containing a bit for every possible UNIBUS run. UNIBUS runs are numbered from 0 to 15, and the system is restricted to a maximum of 16 UNIBUS runs. There are four UNIBUS runs reserved for the maximum of four processors. The numbering allows a maximum of 12 switched buses. However, a switched UNIBUS cannot be connected to another switched UNIBUS. A primary UNIBUS run would contain a processor, its UNIBUS, and the peripherals directly attached to its UNIBUS; and a secondary run would consist of a switched bus and the devices attached to it.

In the I/O data structures for each controller in the multiprocessor system is an associated UNIBUS run mask. The bit set in the URM defines the UNIBUS run to which the controller is attached. In the Executive, there is a table of connectivity masks, one UNIBUS run mask for each processor in the system. The table represents the UNIBUS runs to which each processor is attached. A bit set in the table mask word for a processor indicates that the UNIBUS run is currently associated with that processor.

To ascertain whether a controller is attached to the current processor, the Executive compares the controller URM with the mask for the processor in the connectivity table. If the same bit is set in both words, the controller is attached to the current processor. If a bus is switched from one processor to another, the system need alter only the connectivity masks of the processors affected.

1.5.2 Conditional Fork

The conditional fork routine ($CFORK) is the method by which the Executive distributes I/O requests to devices connected to another processor. In a multiprocessor system, peripheral devices are generally accessible to only one UNIBUS run. Devices that do have dual-access capability are not necessarily accessible from every UNIBUS. The Executive ensures that, when a driver accesses a controller, the driver process executes under control of the processor in whose I/O space the controller registers reside. An exception is the Executive passing control to a driver for special processing of an I/O packet. In this case, the driver is responsible for ensuring that the process executes on the correct CPU. See the discussion of the UC.QUE bit in Section 4.4.4.

The conditional fork routine is necessary because the system allows processors to remain anonymous as far as task execution is concerned. The system does not restrict execution of a user task to the processor associated with a device to which the task directs I/O. Basically it is the driver processes that need to execute on specific processors.

1.5.3 Processor-Specific Functions

When the Executive calls a driver to initiate I/O, the driver may not be executing on the processor associated with the device unit to which I/O is directed. When the driver requests an I/O packet to process, the Executive must ensure that the driver executes on the correct processor because the driver may access the I/O page. Therefore, the Executive routine ($GTPKT) that dequeues an I/O packet for the driver performs a conditional fork. A cell in the fork block for the device unit contains a UNIBUS run mask that defines the processor to which the unit's controller is attached. The conditional fork routine accesses this cell to ascertain what action to take.

The URM of the device to which the I/O request is directed therefore determines whether the driver may execute on the current processor. If the URM of the device intersects the current processor URM, the conditional fork routine returns and the I/O packet is immediately passed to the driver. The driver then normally proceeds to start the proper I/O function. If execution must be continued on another processor, the conditional fork routine performs a fork (that is, calls the $FORK routine). The driver has no indication that it has become a fork process (that is, the action is transparent to the driver).

To ensure that the driver executes on the correct processor, the fork routine does two operations. First, it creates and queues a fork block for the processor on which the driver must execute. Second, it returns to the driver in such a manner as to force the driver to dismiss itself. As soon as possible, the fork processor restarts the driver process executing on the appropriate processor.

For devices that do not have an assigned controller, the system may defer determining whether the driver executes on the current processor. Therefore, for overlapped seek and dual-access devices, the conditional fork routine is entered after the Executive routine that assigns the controller.

1.6 OVERVIEW OF INCORPORATING A USER-WRITTEN DRIVER INTO RSX-11M-PLUS

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

During system generation, you answer questions concerning the types and quantity of peripheral devices on your system. Based on your answers, the system generation software creates the device data base source files. The file SYSTB.MAC contains the data base definitions for all the DIGITAL-supplied devices that were generated with resident data bases. The files xxTAB.MAC, where xx is the device mnemonic, contain the data base definitions for each of the DIGITAL-supplied devices that were generated with loadable data bases. The files xxDRV.MAC, where xx is the device mnemonic, contain the driver code to support the devices. The system generation software assembles and task builds these modules. The resident driver and data base modules are linked into and become a permanent part of the Executive. The loadable driver and data base modules are task built separately for loading into memory after the Executive has been built.

A privileged system task called LOAD is responsible for loading into memory a driver that is not resident. LOAD creates the necessary interrupt control blocks (ICBs) for accessing a driver and establishes the linkage between the data base structures in the system device tables and the driver code being loaded. Another system task called CON initializes the interrupt vectors to point to the ICBs and actually places the devices on-line. CON can also change the vector and CSR address assignments in a device's data base. Another privileged system task called UNLOAD can remove a loadable driver from memory. (Although UNLOAD removes a loadable driver, it does not remove a loadable data base.)

To incorporate a user-written driver into RSX-11M-PLUS, you first create two modules, one in which you define the data base and the other in which you include the driver code itself. You then must integrate your driver data base and driver code modules into the system device tables. If your data base is resident, the linkages that your data base module must satisfy are: (1) the link of the controller table (CTB) list; and (2) the link of the device control block (DCB) list. The linkage for the driver code connects the DCB for the device that your driver supports to the driver dispatch table (DDT). If your driver and data base are loadable, you must supply in your code symbols and labels that LOAD needs. Your device interrupt vectors are initialized and the devices are placed on-line by CON.

Because the data base for a loadable driver can be loadable, the LOAD task also loads a data base. When you load a driver, LOAD checks to see whether a data base is resident for the type of device whose driver is being loaded. If a data base is not resident, LOAD reads the driver symbol definition file to find the start and end of the data base in the driver image. (Thus, if your driver data base is to be loadable, you must have defined its start and end in the data base source code.) Knowing the start and end, LOAD reads the data base from the driver image. LOAD 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, LOAD performs many consistency and validity checks on the data base being loaded.

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

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

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

Whatever type your driver is, you will use the CON task to initialize the device interrupt vectors and place the devices on-line.

Because LOAD provides consistency and validity checks on a data base being loaded, DIGITAL recommends that you make your driver and its data base loadable. (Additional rationale for making your driver fully loadable is given in Section 1.7.) Furthermore, with a loadable driver and loadable data base, you can more easily modify your driver and its data base. You need not rebuild your Executive and privileged tasks. To change the driver code, you need only build a new driver image, unload the current version, and reload the new version. To change the driver data base, you must build a new driver image (which incorporates the modified data base module), rebootstrap your system, and load the new driver which causes the modified data base to be loaded. (You must bootstrap your system to change the data base because UNLOAD does not unload a data base, and because LOAD does not load a data base for a driver if one is currently loaded for that driver.)

Using a loadable driver with a loadable data base saves work in the long term. During debugging, data base inconsistencies are likely to be caught by LOAD. Thus, you prevent many such errors from later creating system problems.

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

1.7 SPR SUPPORT

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

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

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 throughout the system. This unique name allows the Executive to find the correct CTB for the controller type. For example, the RH11/70 controller has the name RH instead of DB, DS, DR, or MM.

A CTB is a static structure created during system generation. Any user-written driver data base, therefore, must have its own CTB. The user-created controller table must also be linked into the system CTB list.

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- or hardware-specific information and accesses the correct information for a unit which its associated controller owns. One KRB exists for each device controller in the configuration. It stores such data as vector and CSR location, status, and UNIBUS run mask.

In a configuration where a device has only one access path to a controller and the 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, then, 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, there must be some way 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 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. When a device has multiple-access paths, the controller-specific information in the KRB is separate from each independent unit context. In a situation where a device accesses alternate controllers, a driver must request the Executive to assign the unit to a specific controller. The unit assignment involves temporarily associating unit context with the KRB of the specific controller. The SCB, then, holds information connecting it to the KRB of the currently assigned controller.

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 device type. For example, there are two device control blocks for the device TT: on a system that supports terminals connected by both DL11 and DZ11 interfaces.

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 device type must be of equal length. (The UCB length is stored in the device control block.) 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, unit status, and error logging information; 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.

The number of status control blocks depends on the processing support in the Executive. 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 one 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 generic data from unit context. Therefore, the driver data base contains the required KRB cells in the status control block.

If the controller allows parallel operations and the Executive supports this feature, there must be one SCB to store context 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.

2.2 DRIVER DISPATCH TABLE (DDT)

The driver dispatch table[1] 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 either in an interrupt control block if the driver is loadable or in the device interrupt vector in the system common area of the Executive if the driver is resident.

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 advance driver features:

- Deallocate buffers and next command (FDX TTDRV)
- Address checking and conversion (queue optimization disk drivers)

[1] 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.

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 make 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 gets 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 either returns information in registers concerning both the packet to be processed and the associated data in order to gain access to the data structures[1] or causes the driver to dismiss itself. (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 sets the GO bit on the device, which physically initiates the I/O operation.

Typically a driver is called at this entry point when there is a packet in 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.

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 must terminate I/O. When such a termination becomes necessary, a task issues an Executive request and the Executive relays the request to the driver by calling it at this entry point.

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 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.

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

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

entry point. Using this facility, a driver can wait for an interrupt but need not hang up if the interrupt never occurs. Thus, no driver should ever stall on a request because a hardware failure prevented an expected interrupt from happening.

2.2.4 Device Power Failure

The Executive calls the power failure entry point when power is restored after a failure any time the controller is busy (that is, when I/O is in progress). Typically, a driver responds to a power failure in the same manner it responds to a timeout. In such cases, the power failure entry point may simply be a return to the caller because recovery will occur by means of the timeout entry point. The driver is called for both controller and unit power failure unless the driver is associated with a common interrupt controller. For common interrupt controllers, the driver is called at this entry point only for unit power failure and is called at a special entry defined in the common interrupt table for controller power failure.

A driver can be called when power is restored regardless of the existence of an outstanding I/O operation. Refer to the description of the U.CTL control flag UC.PWF in Section 4.4.4 for information on unconditional call on power failure.

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. The driver has 60 seconds to perform whatever synchronization it requires before returning to the Executive. In most cases, however, the driver will return immediately.

2.2.6 Device Interrupt Addresses

Control passes to an interrupt 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. For example, a full duplex device such as a terminal will have two interrupt addresses. The interrupt entry differs from an entry point in that the connections between the device and the driver is more direct--the Executive is not involved.

The interrupt addresses are arranged 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. There is no restriction on the number of vectors each controller has, and the number of vectors is implied by the number of addresses in the interrupt address block.

2.3 TYPICAL CONTROL RELATIONSHIPS

This section presents different arrangements of the control structures that are found in RSX-11M-PLUS. 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 you can find in a system. 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 RK05J controller with two RK05 drives attached. 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.

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 Single Controller, Serial Operation

Another typical conventional arrangement of structures for a user-written driver is shown in Figure 2-2, which could represent two LP11 controllers, one with an LP04 and the other with an LP05 attached. It represents the simplest case of driver processing. Figure 2-2 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.

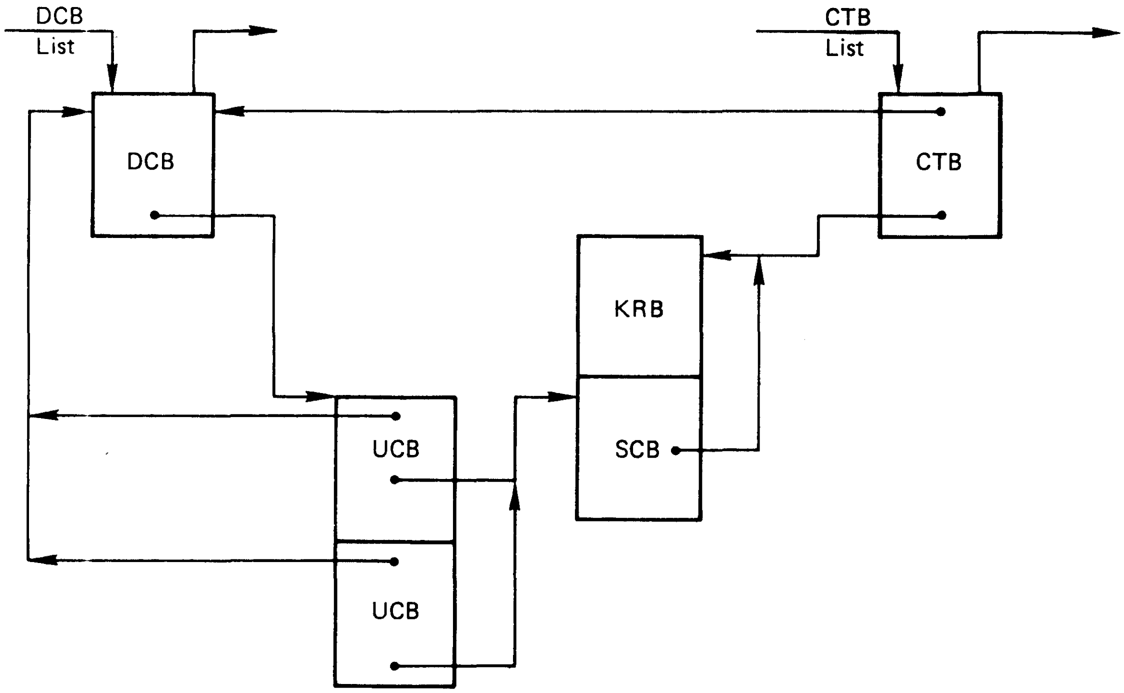


Figure 2-1 Multiple Units per Controller, Serial Unit Operation

2.3.3 Parallel Unit Operation

Some devices, such as the RK06, allow multiple units to have seek operations in progress at the same time. In particular, the RK06 allows such operations to overlap a data operation. 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.

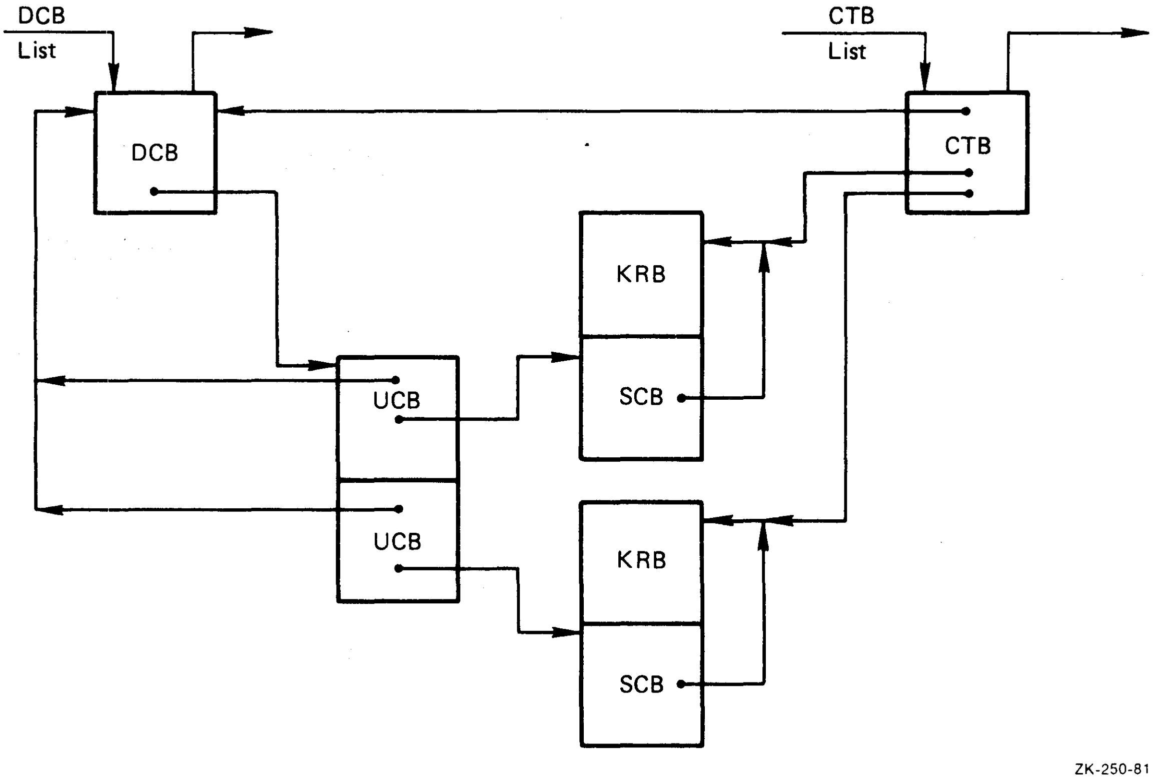


Figure 2-2 Single Controller, Serial Operation

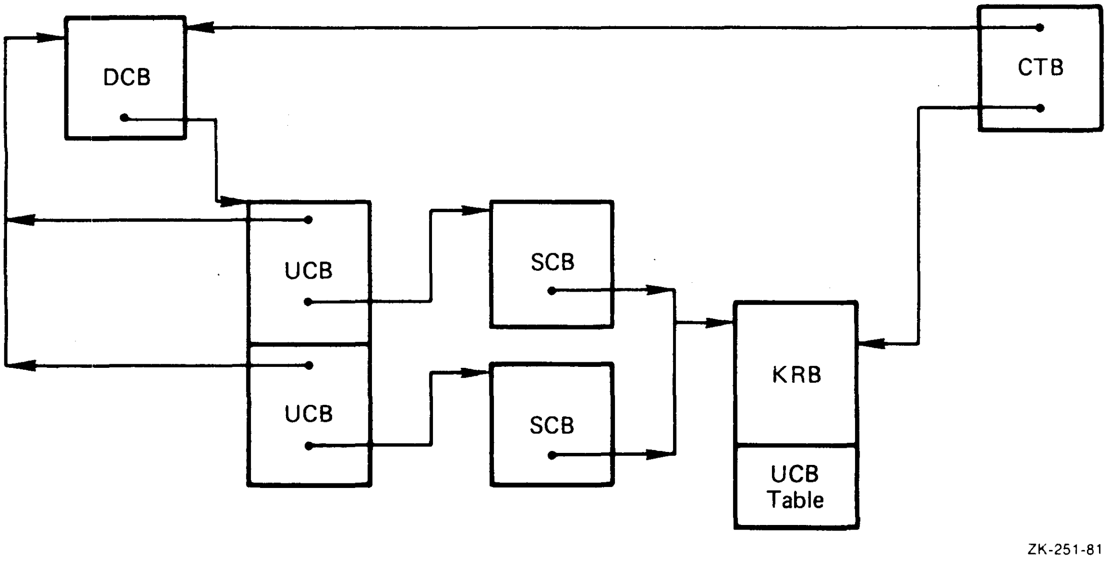


Figure 2-3 Parallel Unit Operation (Overlapped Seek)

2.3.4 Multiple-Access (Dual-Access) Operation

Some devices, such as the RK06, have a dual port option that provides multiple-access paths to units. On the RK06, dual ports on the unit enable a single unit to be electronically switched between two controllers. Figure 2-4 shows the several changes in the structures needed to support dual-access operations.

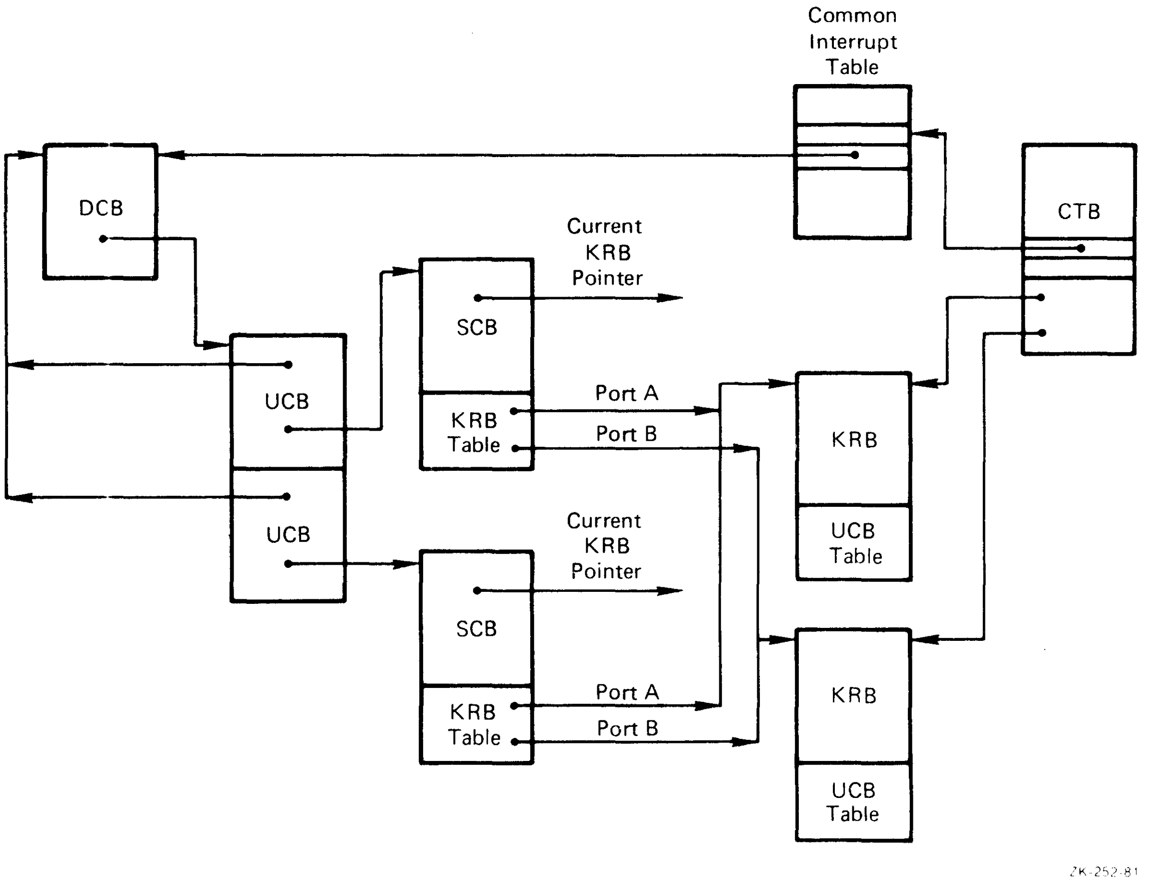


Figure 2-4 Dual-Access Operation

Separate status control blocks are needed for each unit because, if one controller is currently busy, the alternate controller can be idle and allow the operation to proceed. The difference in the dual-access structure is that the SCB no longer points to the same controller request block all the time as in the overlapped seek arrangement. The Executive can change the SCB pointer to a KRB to reflect the capability to electronically switch a unit between two controllers.

To enable the software to differentiate which controllers may access a unit, the SCB has a table of KRB addresses. For dual-access disks, the table contains two entries: the addresses of the controller request blocks for each controller between which the unit can be switched.

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 2-5, 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 loadable 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 tasks such as LOA and UNL reference the data in the PCB. A driver is not concerned with the PCB.

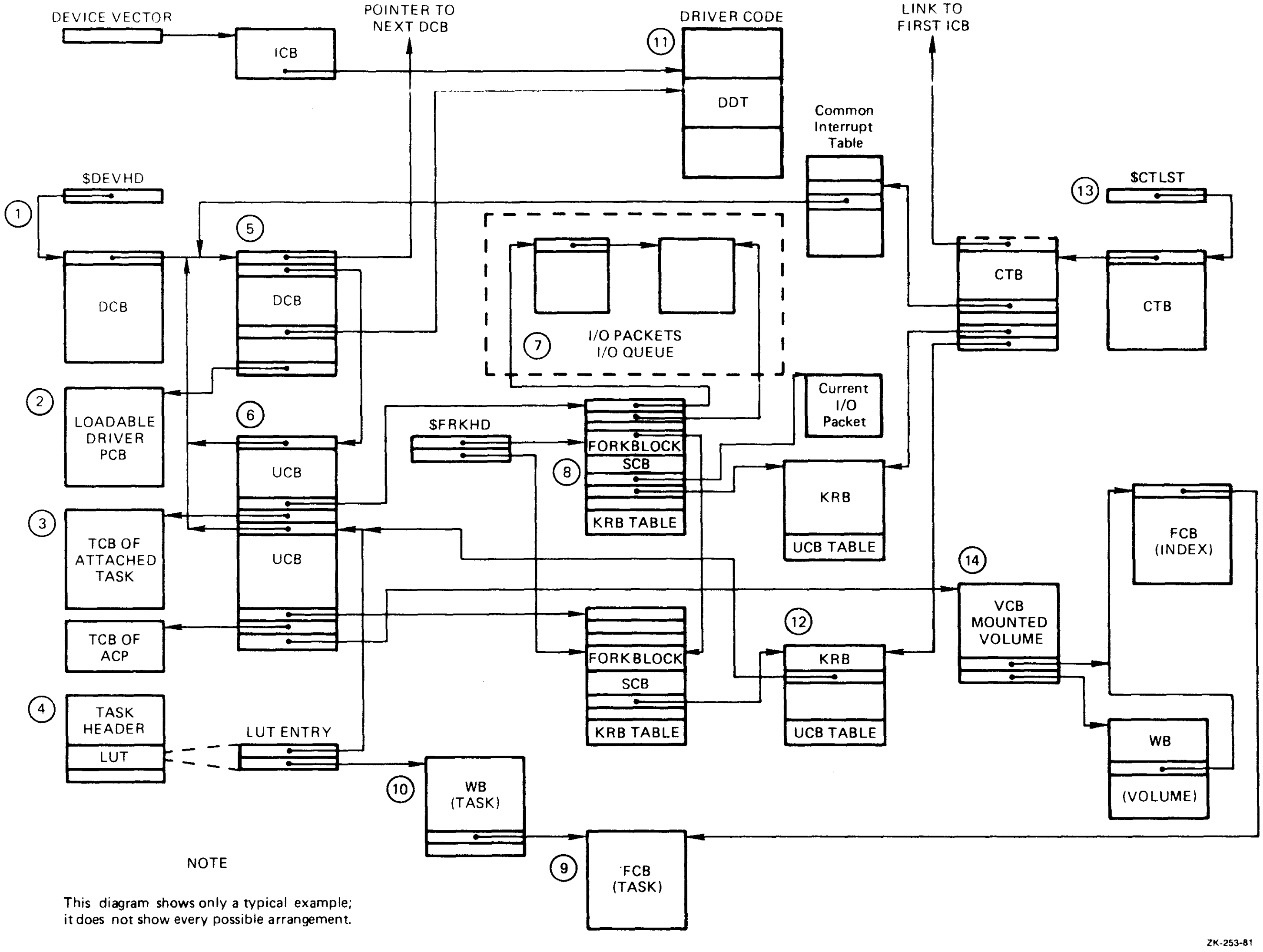
3. If a task is attached to a unit, the UCB has a pointer to the task control block (TCB) of that task.

4. The task header is an independent entity in the I/O data structure and the driver never accesses it. A copy of the task header is taken from the task partition and stored in the Executive dynamic storage region whenever the task is actually in memory. This copy is then used by the Executive.

   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 continuous 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 or a reconfiguration request.



NOTE

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

Figure 2-5 Composite I/O Data Structures

6. Each unit control block contains a pointer back to its related DCB. This backpointer allows the Executive interrupt dispatch code to enter the proper 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.

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 pool). The Executive connects the packets by a pointer in each packet to form a singly-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.

   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.[1]

   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. If multiple-access paths between a unit and controller are possible, the KRB pointer is dynamic. The KRB to which the SCB points at one instant therefore, is considered to be the currently assigned KRB. To reflect the existence of alternate controllers, a table of pointers to all the possible KRBs is contained in the SCB, separate from the pointer to the currently assigned KRB.

   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 processor recovers the proper context from the fork block.

[1] 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.

On multiprocessor systems, the fork block contains an extra cell to define the processor on which the driver must execute the I/O request. The Executive routine that preserves context in the fork block ensures that certain driver code is processed on a particular processor.

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

11. The driver dispatch table (DDT) is part of the driver code and, through the vector and the interrupt control block, is the means by which the device interrupts are passed to 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 table of KRB address in the CTB is static.

    The KRB table 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 Executive determines the correct KRB by indexing the CTB with the controller number from the PS word in the vector.

    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 records (in the attention summary register) 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.

    To support unit operation in parallel, the KRB also contains a queue to regulate controller access. This queue, the controller request queue, is a list of fork blocks for driver processes that have requested and have been denied access to the controller. The driver requests access to a controller. If 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.

The KRB queue cells have 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 must be linked into the list of system-defined CTBs. This list is the mechanism by which system routines, such as those for reconfiguration, 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 does not access 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 RSX-11M-PLUS, 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

   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 4.4.4 for a description of the UC.QUE bit.)

3.1.2 Driver Processing

8. Request work

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

9. 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.

10. 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.

11. 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 initiator entry point and looks for more work (Step 8). This procedure is followed until the driver finds the queue empty, whereupon the driver returns to its caller and the driver process vanishes.

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

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. Interrupt Save ($INTSV)

3. Create Fork Process ($FORK)

4. 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.[1] 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 Interrupt Save ($INTSV)

A driver should not directly call the $INTSV coroutine but should use the INTSV$ macro call. Therefore, if the driver is loadable, it need not call $INTSV and the macro will not generate the call in the driver. (The interrupt save processing is done by either the interrupt control block or the appropriate common interrupt routine in the Executive.) If a driver is resident, the INTSV$ macro call generates the call to the $INTSV coroutine. The coroutine saves code in the driver because the call is shorter than the code to save and restore the conventional registers R4 and R5. More importantly, the $INTSV coroutine gets the driver onto the system stack if it is not already there. The INTSV$ macro is described in more detail in Section 4.3 and the interrupt entry point is described in Section 4.5.

3.2.3 Create Fork Process ($FORK)

Synchronization of access to shared data bases is accomplished by creating a fork process. When a driver needs to access a shared data base, it must do so as a fork process; the driver becomes a fork process by calling $FORK. The SCB contains preallocated storage for a 4- or 5-word fork block. See Section 4.4.5 for a description of the fork block. Section 7.4 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.

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

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

- Store status if an IOSB address was specified
- Set an event flag if one was requested

[1] 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.

- Determine whether a checkpoint request can now be honored
- 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 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 RSX-11M-PLUS 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 RSX-11M-PLUS 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. No registers are available for use unless $INTSV has been called, or the driver explicitly performs save and restore operations. If $INTSV is called, registers R4 and R5 are available; any other registers must be saved and restored. If the driver is to call $INTSV directly, it must do so immediately because $INTSV attempts to retrieve the controller number from the PS.

2. Noninterruptable processing must not exceed 20 instructions, and processing at the priority of the interrupting source must not exceed 500us.

3. Only a fork process should modify system data bases.

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. By following this aspect of good coding practice, you can reduce debugging time when converting an RSX-11M driver to run on RSX-11M-PLUS. Many of the offsets in the RSX-11M SCB differ physically from those in the RSX-11M-PLUS SCB but have the same symbolic values.

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 you make your data base resident, your data base module will be linked separately from the driver code and will be linked to the system device tables module SYSTB.OBJ. (The source code for the SYSTB module is created in UFD [11,10] during system generation.) If your data base is in a separate module and is to be loadable, 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.

The detailed descriptions of the driver data structures are in Section 4.4. A few fields in the structures are conditional on certain features in the Executive. You therefore must use conditional assembly directives and some system-wide symbols that are defined in the Executive assembly prefix file RSXMC.MAC, which is created during system generation.

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 both loadable and resident driver data bases are described in the following subsections.

4.2.1 General Labeling and Ordering of Data Structures

If you are creating a loadable data base, you must specify, for the LOAD 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.

The characters xx represent the 2-character mnemonic of the device that your driver data base supports. If either or both of these labels are not defined, LOAD 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

If the data base for a driver is to be loadable, the LOAD 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.

If the data base for a driver is to be resident, you must define the start of the first (or only) DCB with the global label $USRTB::. This label is required to link the last DCB defined in the SYSTB module with the DCB in your driver data base. If you fail to supply this symbol, the Task Builder will generate an undefined reference error when it builds the Executive.

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 continuous 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 LOAD routines require this label.

If your data base is resident, you must use the CTB macro at the CTB link word L.LNK. The CTB macro automatically generates a global label that provides the linkage between the last CTB defined in the SYSTB module and the CTB defined in your driver tables module. (The definition of the CTB macro is created in the file RSXMC.MAC during system generation.)

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.

You can write driver code for RSX-11M-PLUS that does one of the following:

1. Supports standard functions and runs on RSX-11M-PLUS only.
2. Supports standard functions and is written so that it is compatible with use on both RSX-11M and RSX-11M-PLUS. (This driver needs separate data bases for each system.)
3. Supports advanced features and runs on RSX-11M-PLUS only. (Although Chapter 1 discusses advanced features, this manual does not describe how to program advanced features. Your best guide to utilizing advanced features is to use a DIGITAL-supplied driver as a model.)

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, RSX-11M-PLUS 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,OPT,BUF
```

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.

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 LOAD 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.
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.

(continued on next page)

Table 4-2 (Cont.)
DDT$ Macro Call Arguments

Argument	Meaning
inx	uses an alternate I/O initiation entry point address label instead of the conventional xxINI form. If you specify inx, the macro uses as the only I/O initiation entry point address the location defined by the label xxinx.
ucbsv	is for compatibility with RSX-11M drivers. If you are writing a driver for RSX-11M-PLUS, you should leave this argument blank. As a result, the macro does not allocate the space for the table of UCB addresses. For guidelines on specifying this argument, refer to Section 4.3.4.
NEW	distinguishes between RSX-11M-PLUS and RSX-11M drivers. If you specify this argument (any character except null), the macro generates two cells to hold the controller and unit status change entry point addresses. The referenced driver entry points must be labelled xxKRB: and xxUCB:. If your driver uses these entry points, it cannot be compatible with RSX-11M unless the two routines are conditionalized. If the argument is null, the macro generates code to use the xxPWF entry point for controller and unit on-line and off-line status changes.
OPT	indicates that the driver supports seek optimization. The referenced entry point must be labelled xxCHK:. The routine corresponding to that label should qualify the I/O request and convert it to cylinder track and sector.
BUF	required if the driver performs buffered input and output. The entry point xxDEA: is generated. NOTE RSX-11M drivers implicitly handle controller and unit on-line and off-line status changes as power failures. Although this default operation (enabled by code generated from leaving this argument null) is not optimal for operation on RSX-11M-PLUS, the driver will probably function properly without being changed to include the xxKRB and xxUCB entry points.

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	is for compatibility with RSX-11M drivers. If you are writing a driver for RSX-11M-PLUS, you should leave this argument null. 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 for RSX-11M-PLUS 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 (a gratuitous operation) for a device that has only one UCB per KRB. If your RSX-11M-PLUS driver has multiple units attached to the same controller, you should leave this argument null. The macro therefore generates code to store in the KRB the UCB address of the unit to process.

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 xxINI 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 an 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 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 macro conditionally generates a call to the Executive $INTSV routine based on whether the driver is loadable. The format of the INTSV$ macro call is as follows:

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

The arguments of the call are described in Table 4-4. If the symbol LD$xx (where xx is the device mnemonic) is not defined, the macro generates the call to $INTSV and defines the priority at which the interrupt service routine will run. Not defining LD$xx indicates that the driver is resident. (For loadable drivers, the interrupt service routine in the Executive dispatches the interrupt.) For both loadable and resident drivers, however, the macro generates the code to load R5 upon an interrupt.

Table 4-4
INTSV$ Macro Call Arguments

Argument	Meaning
dev	is the 2-character device mnemonic.
pri	is the processor priority (PR4, PR5 or PR6) at which the device runs and at which the $INTSV coroutine will run.
nctrlr	is the number of controllers that the driver services (counting from 1).
pswsv	is for compatibility with RSX-11M drivers. If you are writing an RSX-11M-PLUS driver, leave this argument null. If your driver is an RSX-11M driver, this argument has no effect.
ucbsv	is for compatibility with RSX-11M drivers. If you are writing a driver for RSX-11M-PLUS, you should leave this argument null. The macro generates code which uses the controller index returned in R4 by $INTSV, calculates the KRB of the interrupting controller, and loads the UCB address of the interrupting unit into R5. For guidelines on specifying this argument, refer to Section 4.3.4.

4.3.4 Usage of UCBSV Argument in Macro Calls

The DDT$, GTPKT$, and INTSV$ macro calls allow you to specify an argument (ucbsv) that maintains compatibility with RSX-11M drivers. RSX-11M-PLUS does not need to utilize the ucbsv argument. 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 storage is the CNTBL area used by RSX-11M drivers to contain the address of the unit control block of the interrupting devices for each controller. Both the GTPKT$ and INTSV$ macro calls may use this same area. For more information concerning CNTBL, consult the RSX-11M Guide to Writing an I/O Driver.

If you specify the argument ucbsv in the GTPKT$ macro call, it must be the same label that 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 Specifying a Loadable Driver

To specify that a driver is loadable and to enable generation of conditional code, you must define the symbol LD$xx. The definition can appear in either the driver source code or the assembly prefix file RSXMC.MAC. It is usually more convenient to define the symbol in the driver source code because you probably will not have cause to edit RSXMC.MAC. When the symbol is defined, the INTSV$ macro does not generate the call to $INTSV.

4.3.6 Loadable Driver Entry Points for LOAD and UNLOAD

A loadable driver that requires additional initialization and completion functions can define two entry points by labels of the form $xxLOA and $xxUNL (where xx is the 2-character device mnemonic). 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 LOAD routines check for the $xxLOA entry point.

NOTE

> The LOAD routines can perform this function only from MCR. If you attempt to load a driver that has the $xxLOA entry point from VMR, the load operation is terminated with the error message DRIVER REQUIRES RUNNING SYSTEM FOR LOAD/UNLOAD.

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

The driver may use all the registers. When you unload the driver, the UNLOAD routine calls it at the $xxUNL entry point with the same conditions.

These two entry points in the loadable driver are independent of the controller and unit status change entry points used by Executive reconfiguration software. That is, the two entry points $xxLOA and $xxUNL are used for initialization and completion at LOAD and UNLOAD time and not at on-line and off-line status change time.

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.

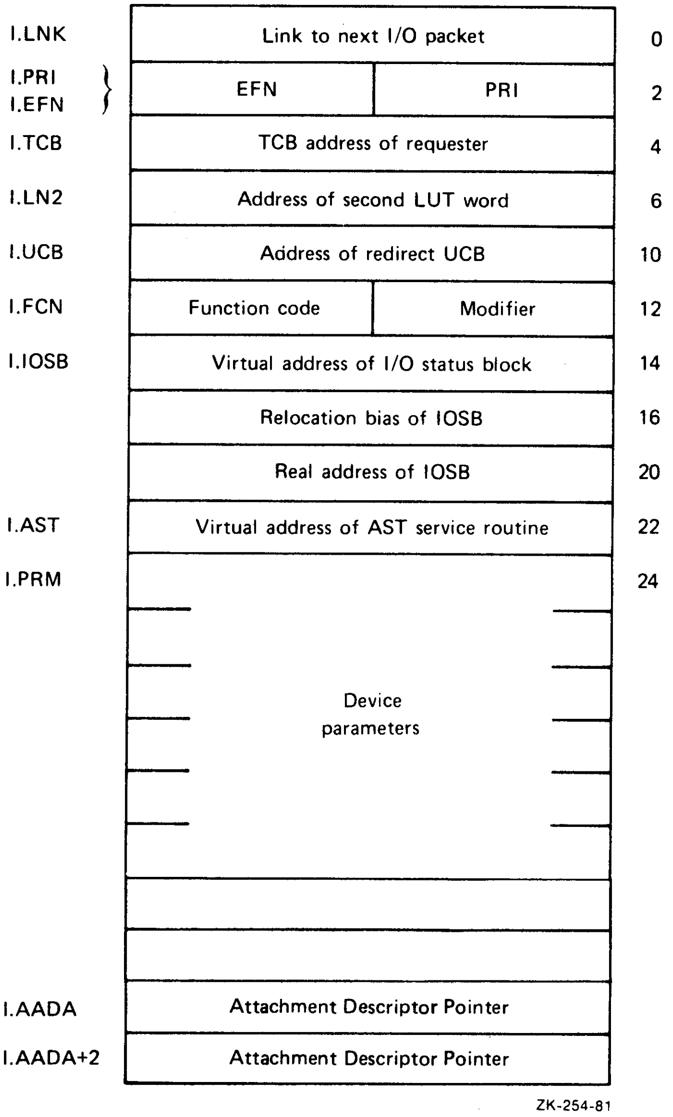
<not initialized, read-write>
Either the driver or some other agent establishes this field, and the driver may read it or write over it.

<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 the I/O Packet, which is constructed and placed in the driver I/O queue by QIO directive processing, and is subsequently delivered to the driver by a call to $GTPKT. The DPB from which the I/O Packet is generated is illustrated in Section 4.4.2.



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		24
I.AADA	Attachment Descriptor Pointer		
I.AADA+2	Attachment Descriptor Pointer		

ZK-254-81

Figure 4-1: I/O Packet Format

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.

I.PRI

Driver access:

Not referenced.

Description:

Priority copied from the TCB of the requesting task.

I.TCB

Driver access:

Not referenced usually. Sometimes referenced at I/O cancel and power failure.

Description:

TCB address of the requesting task.

I.LN2

Driver access:

Not referenced.

Description:

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

I.UCB

Driver access:

Not referenced by conventional driver; frequently referenced by full duplex drivers.

Description:

Contains the address of the unit to which I/O is to be directed. I.UCB is the address of the Redirect UCB if the starting UCB has been subject to an MCR Redirect command. 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 bits. During predriver initiation the Executive compares the function code with a function mask value in the DCB. The driver interprets the modifier bits.

I.IOSB

Driver access:

Not referenced.

Description:

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

I.IOSB+2 and I.IOSB+4 contain the address doubleword for the IOSB (see Section 7.2 for a detailed description of the address doubleword). The first word contains the relocation bias of the IOSB; the bias is, in effect, the number of the 32-word block in which the IOSB starts.

The second word is formatted as follows:

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

The displacement in block is the offset from the block base. The value 6 in bits 13 through 15 is constant. It is used to cause an address reference through Kernel Address Page Register 6 (APR6).

Discussion of the address doubleword is deferred to Section 7.3 because you seldom have to be concerned with its contents or format in writing a conventional driver. Its construction and subsequent manipulation are normally external to the driver. Subroutines are provided as Executive services for programmed I/O to render the manipulations of I/O transfers transparent to the driver itself.

I.AST

Driver access:

Not referenced.

Description:

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

I.PRM

Driver access:

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 could be an Executive address, 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 4-2. 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):

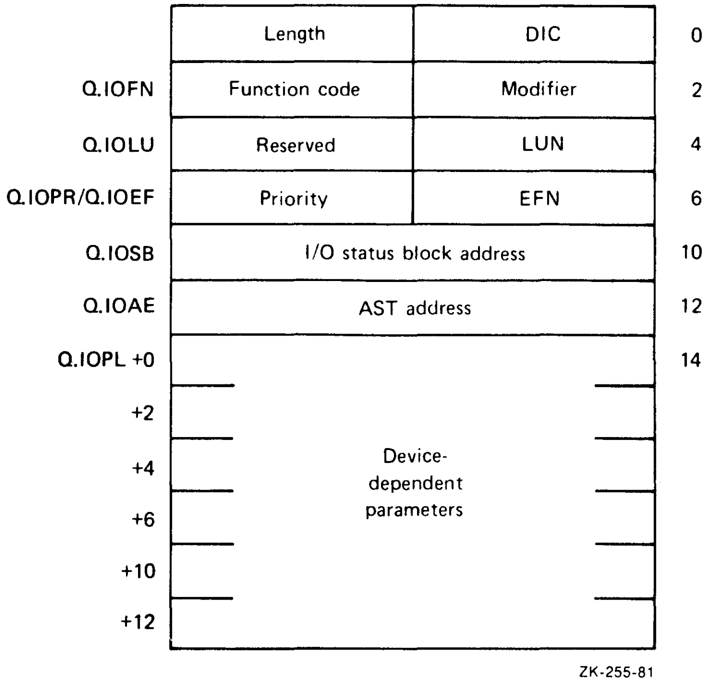
The length of the DPB, which for the RSX-11M and RSX-11M-PLUS QIO directive is always fixed at 12 words.

DIC (required):

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

Q.IOFN (required):

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



	Length	DIC	0
Q.IOFN	Function code	Modifier	2
Q.IOLU	Reserved	LUN	4
Q.IOPR/Q.IOEF	Priority	EFN	6
Q.IOSB	I/O status block address		10
Q.IOAE	AST address		12
Q.IOPL +0	Device-dependent parameters		14
+2			
+4			
+6			
+10			
+12			

Figure 4-2: 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 RSX-11M-PLUS, 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 4-3 is a schematic layout of the DCB. The DCB describes the static characteristics of a device controller and the units attached to the controller. All fields must be specified.

The fields[1] in the DCB are described as follows:

D.LNK (link to next DCB)

Driver access:

Initialized, not referenced.

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

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.

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

ZK-256-81

Figure 4-3: Device Control Block

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

D.UCBL (UCB length)

Driver access:

Initialized, not referenced.

Description:

The 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:

Initialized, 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.) For a resident driver, your code references the symbol $xxTBL, which is generated by the DDT$ macro to mark the start of the driver dispatch table. For a loadable driver, then, you should initialize this field to zero, which indicates that the driver is not in memory.

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.[1] Four masks, with two words per mask, are described by the bit configurations that you establish for these words:

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

[1] 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 template. For example, the RK11 driver (DKDRV) is one that does not use advanced features.

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). 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 RSX-11M-PLUS 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 RSX-11M-PLUS system interpretation. DIGITAL suggests that functions having an RSX-11M-PLUS system counterpart use the RSX-11M-PLUS 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 8.3 for an example of a driver that does this. (Buffer addresses in standard format are automatically converted to Address Doubleword format.)

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

4. Create the No-op Function mask by deciding which legal functions are to be no-op. 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 code 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 whether a driver is loadable or resident and, if loadable, whether it is in memory. Zero and nonzero values for these two pointers have the meanings shown in Figure 4-4.

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

ZK-223-81

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

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.

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 RSX-11M/M-PLUS I/O Drivers Reference Manual for a description of standard I/O functions.) If QIO directive processing encounters a function code of 3 or 4 and the code is not no-op, 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 Tables 4-6, 4-7 and 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		Unused
31	100000		Unused

Table 4-6
Mask Word Bit Settings for Disk Drives

Bit	RSX-11M-PLUS	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

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

Bit	RSX-11M-PLUS	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		
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	RSX-11M-PLUS	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 4-5 is a layout of the UCB (a variable-length control block). One UCB exists for each physical device-unit generated into a system configuration. For user-added drivers, this control block is defined as part of the source code for the driver data structure.

The fields[1] in the UCB are described below:

U.UAB (0)

Driver access:

Initialized, not referenced.

Description:

For terminal UCBs only. It is required only if accounting support is on the system (A$$CNT is defined) but may be present if accounting support is not on the system. This value is used to access the user accounting block in secondary pool.

U.MUP

Driver access:

Not initialized, not referenced.

Description:

For terminal UCBs only. Bits 1 to 4 contain an index to a table which contains the address of CLI Parser Block (CPB) for the current CLI; the remaining bits are used for other terminal specific features and are defined as follows:

UM.OVR	Override CLI indicator
UM.CLI	CLI indicator
UM.DSB	Terminal diabled because CLI eliminated.
UM.NBR	No broadcast
UM.CNT	Continuation of command line in progress
UM.CMO	Command is in progress from this terminal
UM.SER	Terminal is in serial mode
UM.KIL	TTDRV should tell MCR to flush all pieces of a continued command if the user types CTRL/C.

U.LUIC

Driver access:

Not 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. This word is altered by logging into the system.

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

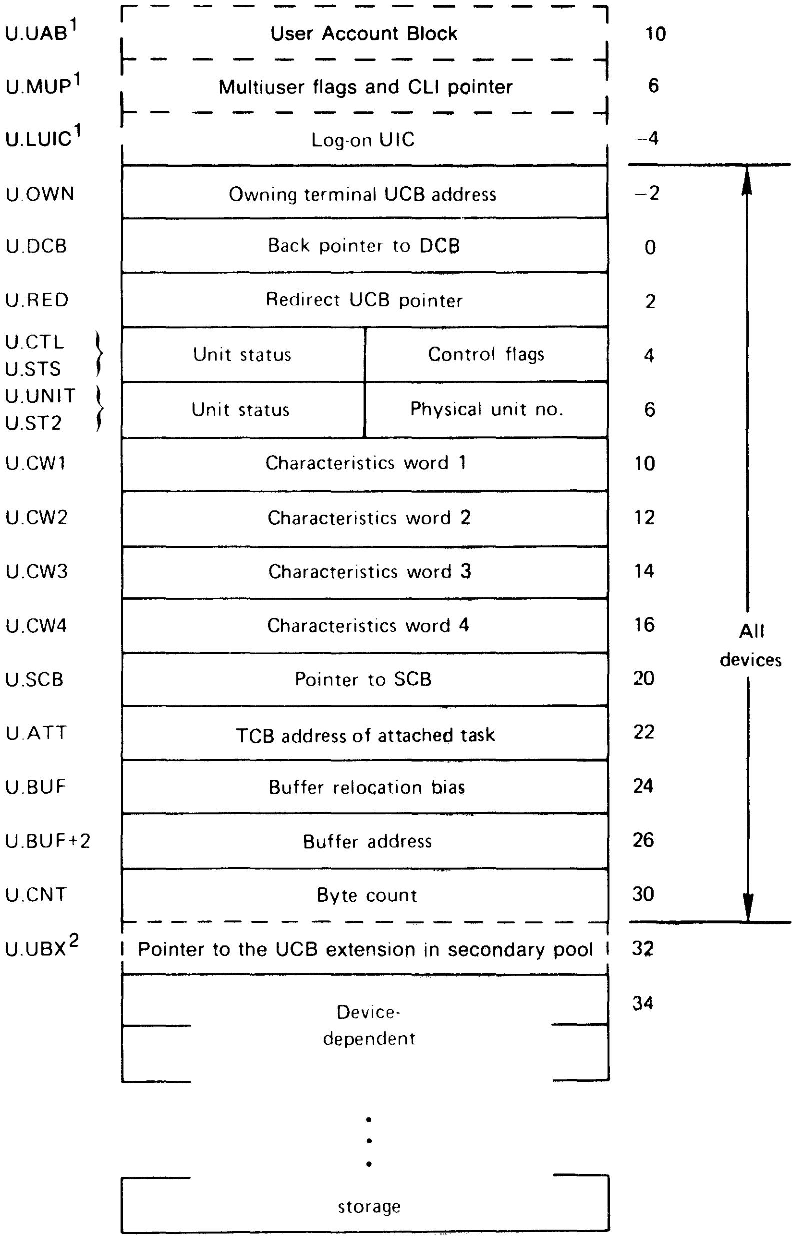
U.OWN (0)

Driver access:

Initialized, not referenced.

Description:

Only in multiuser systems: the UCB address of the owning terminal for allocated devices.



U.UAB[1]	User Account Block		10	
U.MUP[1]	Multiuser flags and CLI pointer		6	
U.LUIC[1]	Log-on UIC		–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.UBX[2]	Pointer to the UCB extension in secondary pool		32	
	Device-dependent ⋮ storage		34	

1. This offset appears only for terminal devices (that is, devices that have DV.TTY set) in multiuser systems.

2. This offset appears only for those devices that have DV.MSD set.

ZK-257-81

Figure 4-5: 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. This field is updated as the result of an MCR Redirect command. The redirect chain ends when this word points to the beginning of the UCB itself (U.DCB of the UCB, to be precise).

U.CTL (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 4-6 shows the layout of the unit control byte.

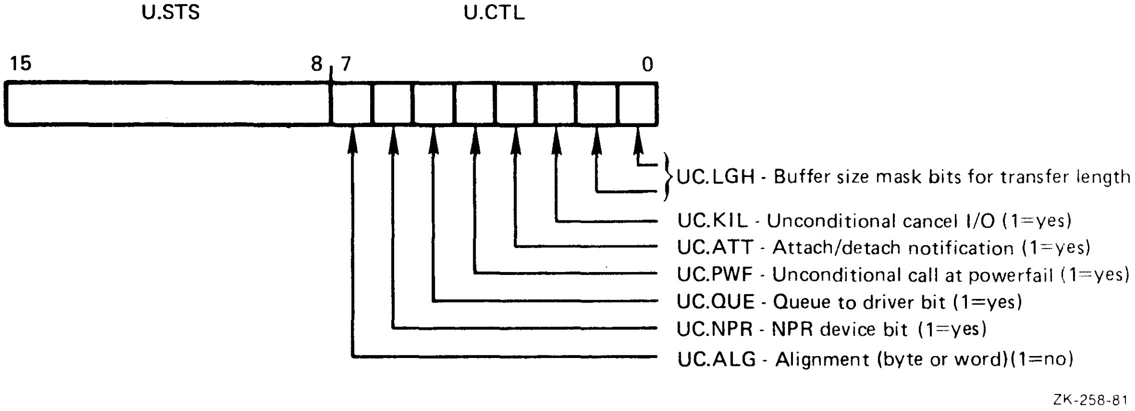

U.STS
U.CTL
15
8
7
0
UC.LGH - Buffer size mask bits for transfer length
UC.KIL - Unconditional cancel I/O (1=yes)
UC.ATT - Attach/detach notification (1=yes)
UC.PWF - Unconditional call at powerfail (1=yes)
UC.QUE - Queue to driver bit (1=yes)
UC.NPR - NPR device bit (1=yes)
UC.ALG - Alignment (byte or word)(1=no)
ZK-258-81

Figure 4-6: 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. If UC.ALG is 1, then buffers must be word-aligned.

UC.ATT - Attach/Detach notification.

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

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

If set, the driver is called on a Cancel I/O request, even if the unit specified is not busy. Typically, the driver is called on Cancel I/O only if an I/O operation is in progress. 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 8.3 for an example of a driver that does this.

On multiprocessor systems, certain restrictions apply to this form of I/O processing. No driver should process an I/O packet received directly from the QIO processor without first performing a conditional fork operation (that is, call $CFORK) to guarantee execution on the correct processor. Unless the driver is running on the correct processor, it must not process a packet that causes access to the device registers. The restriction does not apply if the driver merely uses the current task context to map secondary I/O buffers and then queues the I/O packet itself. In summary, packets received directly from $DRQIO may not be processed directly unless they cause no activity on the I/O page (and thereby do not need to be executed on a particular processor) or unless an intervening call to $CFORK has been performed.

UC.PWF - Unconditional call on power failure bit.

If set and the unit is on-line, the driver is always to be called when power is restored after a power failure occurs. Typically, the driver is called on power restoration only when an I/O operation is in progress. See the discussion in Sections 4.3.6 and 4.5 of the entry points in the DDT for LOAD and UNLOAD and for controller and unit status change.

UC.NPR - NPR device bit.

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

UC.LGH - Buffer size mask bits (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.

NOTE

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

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 4-7 shows the layout of the unit status byte.

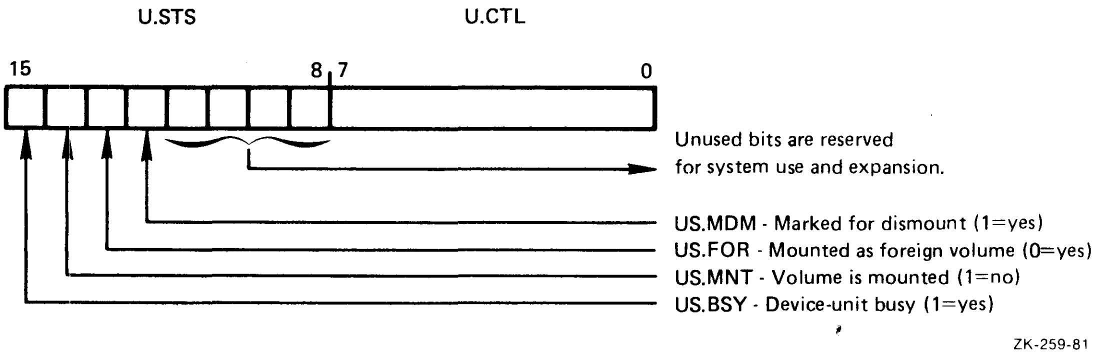

U.STS
U.CTL
15
8
7
0
Unused bits are reserved for system use and expansion.
US.MDM - Marked for dismount (1=yes)
US.FOR - Mounted as foreign volume (0=yes)
US.MNT - Volume is mounted (1=no)
US.BSY - Device-unit busy (1=yes)
ZK-259-81

Figure 4-7: Unit Status Byte

US.MDM, US.MNT, and US.FOR apply only to mountable devices.[1]

[1] If your user-written driver services a mountable device, refer to Section 4.5.9 for information on volume valid processing.

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 set, device is marked for dismount.

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 4-8.

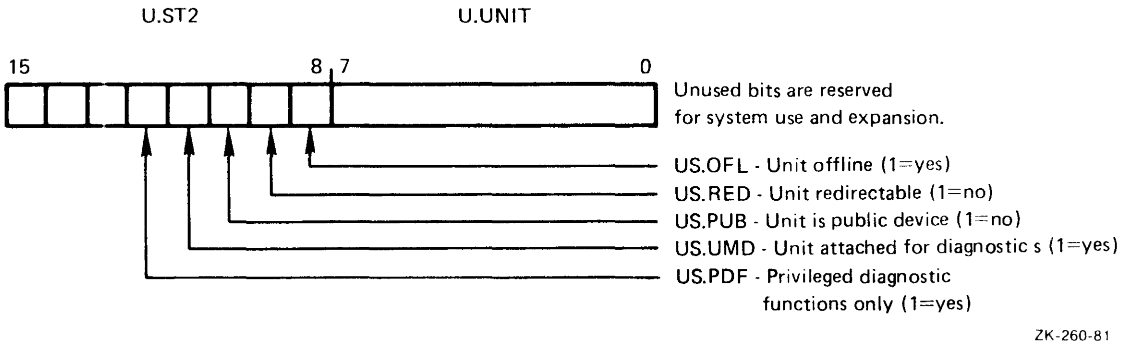

U.ST2
U.UNIT
15
8
7
0
Unused bits are reserved
for system use and expansion.
US.OFL - Unit offline (1=yes)
US.RED - Unit redirectable (1=no)
US.PUB - Unit is public device (1=no)
US.UMD - Unit attached for diagnostic s (1=yes)
US.PDF - Privileged diagnostic
functions only (1=yes)
ZK-260-81

Figure 4-8: Unit Status Extension 2

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.

U.CW1 (device-specific characteristics)

Driver access:

Initialized, not referenced.

Description:

The first of a 4-word continuous 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[1]

U.CW2 (device-specific characteristics)

Driver access:

Initialized, read-write.

Description:

Specific to a given device driver (available for working storage or constants).[2]

[1] If your user-written driver services a mountable device, refer to Section 4.5.9 for information on volume valid processing.

[2] 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.

U.CW3 (device-specific characteristics)

Driver access:

Initialized, read-write.

Description:

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

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.

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.

[1] 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:

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 identical to that for the second two words of I.IOSB in the I/O Packet. See Section 4.4.1 for a description of I.IOSB in the I/O packet.

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

For NPR device drivers, these two words represent what the driver uses to initiate the transfer operation. For both UNIBUS and MASSBUS NPR devices, word 2 contains the low-order 16 bits of the physical address. For a UNIBUS NPR device, bits 4 and 5 in word 1 are memory extension bits; for a MASSBUS NPR device (the KS.MBC bit is set), bits 0 through 5 are the memory extension bits. It is the driver's responsibility to set the function code, interrupt enable, and go bits. This action must be accomplished by a Bit Set (BIS) operation so that the extension bits are not disturbed. The driver must move these words into the device control registers to initiate the I/O operation.

For a typical UNIBUS NPR device driver, the word layout is as follows:

Word 1

Bit 0	Go bit initially set to zero
Bits 1,2,3	Function code--set to zeros
Bits 4,5	Memory extension bits
Bits 6	Interrupt enable--set to zero
Bits 7 through 15	Zero

Word 2

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

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

The details of the construction of the Address Doubleword appear in Section 7.2.

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:

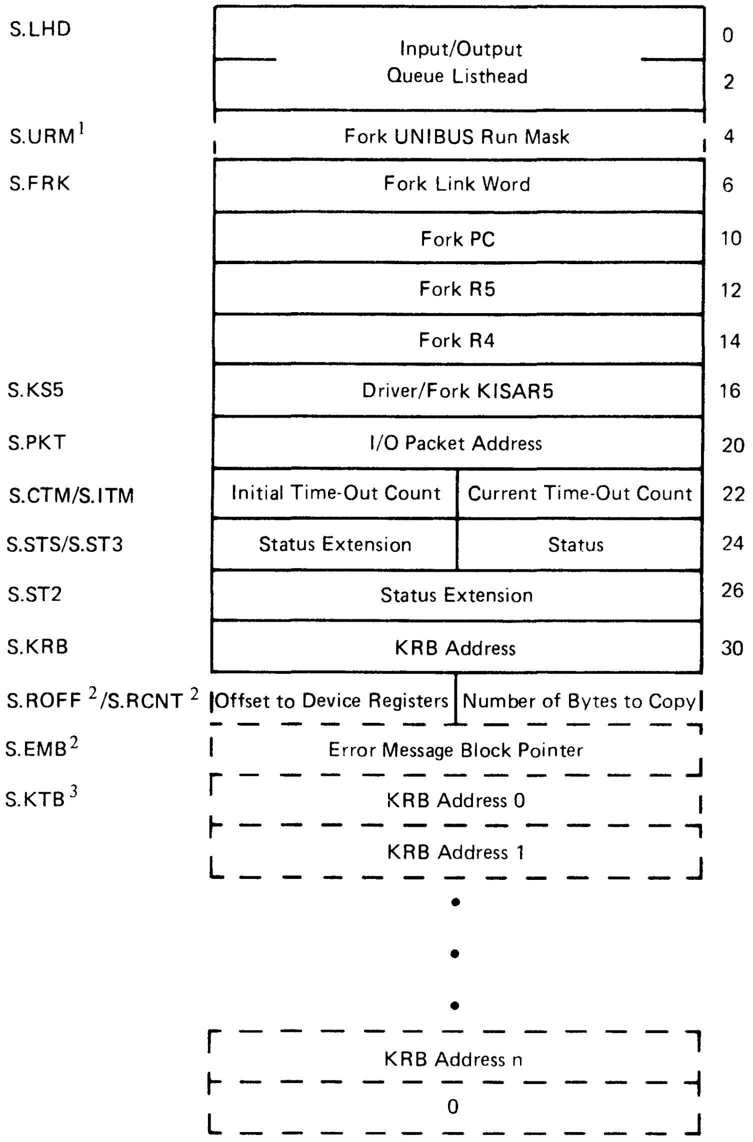
The driver establishes this variable-length block of words to suit device-specific requirements. For example, a disk driver uses the first words to store the disk geometry as follows:

```
.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 4-9 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.



If the symbols below are defined at system generation time, the related cells marked with a number appear in the structure.

[1] Multiprocessor support (M$$PRO)

[2] Appears only if driver supports error logging

[3] If the system has multiaccess device support (M$$ACD) and the driver is multiaccess (S2.MAD)

ZK-261-81

Figure 4-9: Status Control Block

The fields[1] in the SCB are described as follows:

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

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.URM (controller UNIBUS run mask)

Driver access:

Initialized, not referenced.

Description:

This word appears only in a multiprocessor system (that is, M$$PRO is defined). It contains a UNIBUS run mask that defines the UNIBUS run to which the currently assigned controller is attached. When controller assignment is made, this cell is set from K.URM. For the purposes of running a driver on the correct processor, S.URM is used exclusively and independently of the value of S.KRB or K.URM. If S.KRB is not equal to zero, and if S.URM is not equal to K.URM (an unusual situation), then the driver must properly handle the fact that it will run on a different processor from the one its currently assigned KRB would normally warrant. It is possible that the processor on which the driver will run has the CSRs at a different location from that stored in the current KRB. ADJACENCY WITH THE FORK BLOCK IS ASSUMED!

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. The Fork processor also depends on the adjacency of S.URM and S.KS5 if the required support is generated into the system.

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 has no meaning for a driver that is not loadable. It is set by LOAD, 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)

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:

RSX-11M-PLUS 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 4-10 shows the layout of this byte.

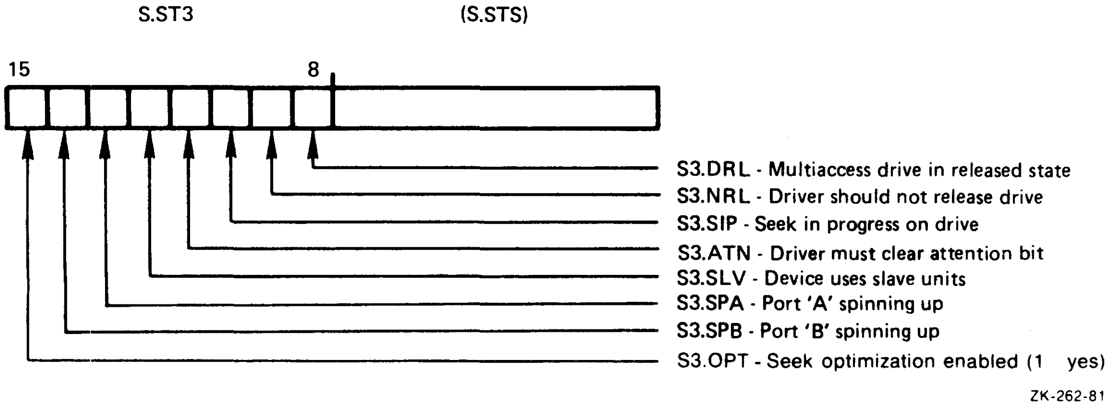

S.ST3
(S.STS)
15
8
S3.DRL - Multiaccess drive in released state
S3.NRL - Driver should not release drive
S3.SIP - Seek in progress on drive
S3.ATN - Driver must clear attention bit
S3.SLV - Device uses slave units
S3.SPA - Port 'A' spinning up
S3.SPB - Port 'B' spinning up
S3.OPT - Seek optimization enabled (1 yes)
ZK-262-81

Figure 4-10: 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.DRL=1

If this bit is set, the drive is in the released state. Drivers that support dual-access (dual-port) operation set this bit after completion of the release command by the drive. The Executive routines Request Controller for Control Function ($RQCNC) and Request Controller for Data Transfer ($RQCND) test this bit to decide whether the drive is in a released state and whether the Executive should attempt load balancing by switching ports.

S3.NRL=2

If this bit is set, a driver does not release the drive. This bit exists solely for DIGITAL to maintain the device and to debug the driver. Drivers that support dual-access (dual-port) operation examine this bit and, if it is set, do not issue the release command to the drive and do not set the S3.DRL bit. If this bit is set, reconfiguration dual-port activity (that is, port on-line and off-line operations) will not function properly.

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 but does support error logging (that is, cassette and magtape), this bit is set to indicate that a positioning operation is in progress.

S3.ATN=10

This bit is used only by MASSBUS devices. The Executive common interrupt module DVINT checks this bit; if it is set, then the driver must clear the attention bit in the Attention Summary Register. If this bit is not set, DVINT itself clears the attention bit in the Attention Summary Register.

S3.SLV=20

If this bit is set, the device connects to slave units. Certain devices, such as magnetic tape controllers attached to a MASSBUS controller, can in turn have units attached to them. These units are referred to as slave units. Thus, if this bit is set, the SCB describes a tape controller to which slave units can be attached.

S3.SPA=40

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

S3.SPB=100

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

S3.OPT=200

If this bit is set, seek optimization is enabled for this device. $GTPKT uses this bit to determine whether optimization is to be used. An MCR SET command can set and clear this bit. If you select seek optimization support for a Digital-supplied device during system generation, SYSGEN sets this bit in that device SCB when it creates the device data base structures.

S.ST2 (controller status extension)

Driver access:

Initialized.

Description:

This status word defines certain status conditions for the controller-unit combination. Figure 4-11 shows the layout of this word.

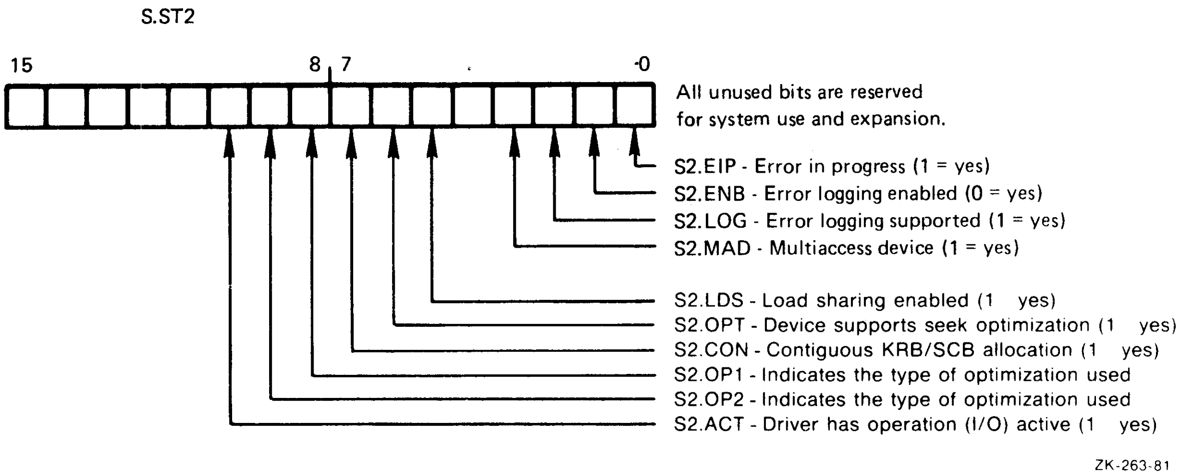

S.ST2
15
8
7
0
All unused bits are reserved for system use and expansion.
S2.EIP - Error in progress (1 = yes)
S2.ENB - Error logging enabled (0 = yes)
S2.LOG - Error logging supported (1 = yes)
S2.MAD - Multiaccess device (1 = yes)
S2.LDS - Load sharing enabled (1 yes)
S2.OPT - Device supports seek optimization (1 yes)
S2.CON - Contiguous KRB/SCB allocation (1 yes)
S2.OP1 - Indicates the type of optimization used
S2.OP2 - Indicates the type of optimization used
S2.ACT - Driver has operation (I/O) active (1 yes)
ZK-263-81

Figure 4-11: 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.EIP=1

This bit is reserved for DIGITAL error logging routines.

S2.ENB=2

This bit is reserved for DIGITAL error logging routines.

S2.LOG=4

This bit is reserved for DIGITAL error logging routines.

S2.MAD=10

This bit indicates the presence of the table of KRB addresses at the end of the Status Control Block. If this bit is set, the device is a multiaccess device and the SCB has a KRB table containing pointers to the KRBs of the controllers capable of accessing the device.

S2.LDS=40

This bit enables and disables load sharing for dual-access devices. If this bit is set, the Executive may dynamically switch ports and therefore alter controller assignment when establishing an access path for a driver. If this bit is not enabled, the Executive does not alter the current controller assignment. This feature permits static controller assignment, perhaps for diagnostic operations.

Devices (such as terminals) with S2.LDS clear have drivers that explicitly manage controller assignment.

S2.OPT=100

If this bit is set, this device supports queue optimization. This bit, used by $DRQRQ, determines whether to call the block check and convert the LBN routine in the driver.

S2.CON=200

This bit indicates the continuous 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.OP1=400
S2.OP2=1000

These bits indicate the type of optimization selected for this device. An MCR command can set and clear these bits. These two bits give you three options of queue optimization. They are as follows:

S2.OP2,S2.OP1 = 0,0	Nearest cylinder
S2.OP2,S2.OP1 = 0,1	Elevator
S2.OP2,S2.OP1 = 1,0	CSCAN
S2.OP2,S2.OP1 = 1,1	Reserved

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. For non-multiaccess devices, it is set during system generation and never altered. For multiaccess devices with load-sharing enabled, it may take on the value of one of the KRB pointers in the KRB table, S.KTB. 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. A device which has no KRB will not have S2.CON set.

The first cell in the KRB (K.CSR) contains the control and status register (CSR) address for the controller. The offset K.CSR will always be zero so that the pointer (S.KRB) will always connect directly to the cell containing the CSR address.

S.ROFF This byte is reserved for devices that support DIGITAL error logging software. This value is an offset from S.CSR/K.CSR to indicate the start of the device registers. It is typically zero.

S.RCNT This byte is reserved for devices that support DIGITAL error logging software. It represents the minimum number of words of I/O page registers that this device has.

S.EMB This word is reserved for devices that support DIGITAL error logging software.

S.KTB (KRB addresses)

Driver access:

Initialized, not referenced.

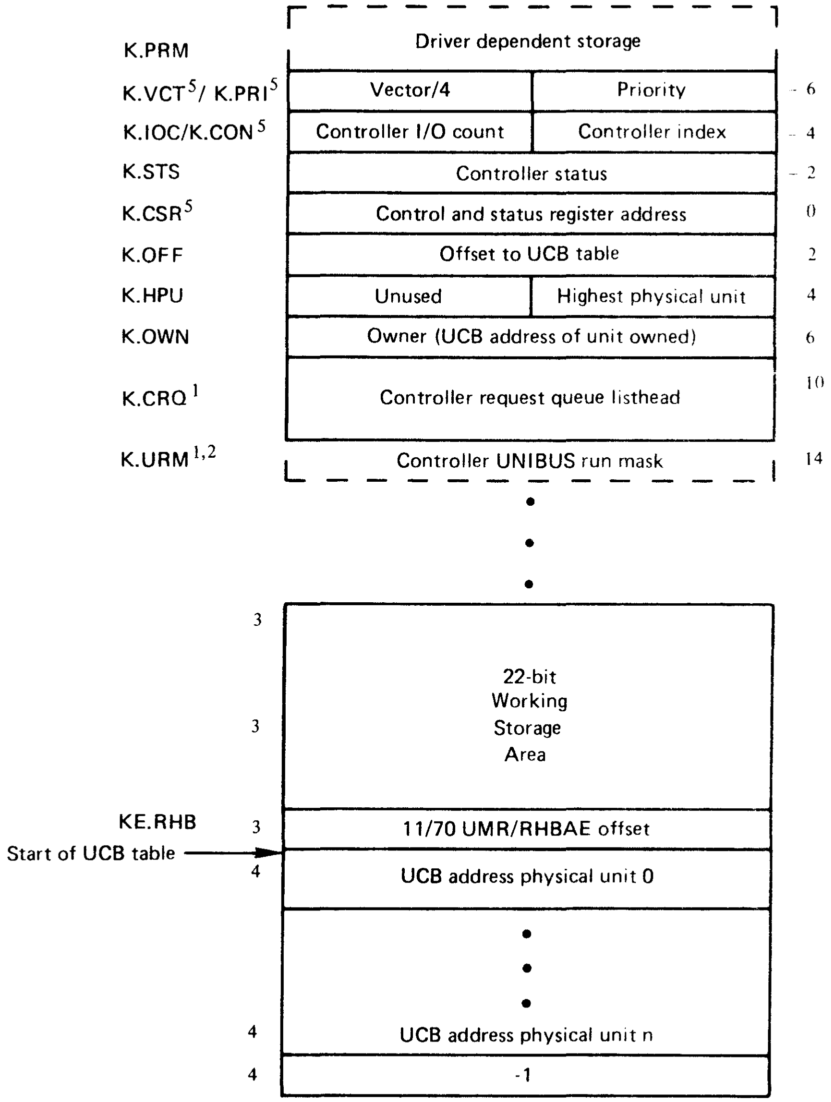
Description:

This table appears only if the system has multiaccess device support (M$$ACD is defined) 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. For devices with executive load sharing supported (S2.LDS set), bit zero of each word is an on-line and off-line flag which, when set, indicates that KRB is off-line with respect to this SCB and should not be considered for controller assignment. Devices with S2.LDS clear have drivers that explicitly manage controller assignment. 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. When drivers explicitly manage controller assignment, 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 4-12 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 continuous space. With such continuous 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 continuous SCB/KRB allocation.



[1] If cont guous allocation of KRB and SCB is used (that is, if S2.CON is set), this field overlaps the I/O request queue.

[2] This fit ld is for multiprocessor support (M$$PRO is defined).

[3] This ar:a is for 11/70 extended memory support (M$$EXT is defined).
The art a extends in a negative direction from the start of the UCB Table.

[4] If KS.L CB is set, then this table appears (allows overlapped function interrupts).

[5] The S.>xx forms of these offsets are valid only for devices that perform a single operation on a controller at a time. For such devices, S2.CON is set and the SCB and KRB are allocated in a contiguous area. See Figjre 4-7 for the contiguous SCB/KRB structure.

ZK-264-81

Figure 4-12: Controller Request Block

The fields[1] in the KRB are described as follows:

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

K.PRM (device-dependent storage)

Driver access:

Initialized, read-write.

Description:

LOAD does not relocate any addresses in this area.

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 (refer to the sample data base in Chapter 8 and to the listing of the HWDDF$ macro).

K.VCT (interrupt vector divided by 4)

Driver access:

Initialized, not referenced.

Description:

Interrupt vector address divided by 4. Because you can use the CON task to change the vector value, you need not be overly concerned with initializing K.VCT to the correct value. If K.VCT equals zero, then neither LOAD nor UNLOAD takes any vector action. In particular, LOAD does not create any interrupt control block linkage for this KRB.

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:

This is an I/O count used by the system to keep track of how busy the controller is. The value is related to the number of outstanding requests queued for this controller. This is a weighted number to be used only by the system to judge the relative activity of one controller with respect to another.

K.STS (controller-specific status)

Driver access:

Initialized, not referenced.

Description:

This word is used as a status word that concerns the controller. Figure 4-13 shows the layout of the controller status word.

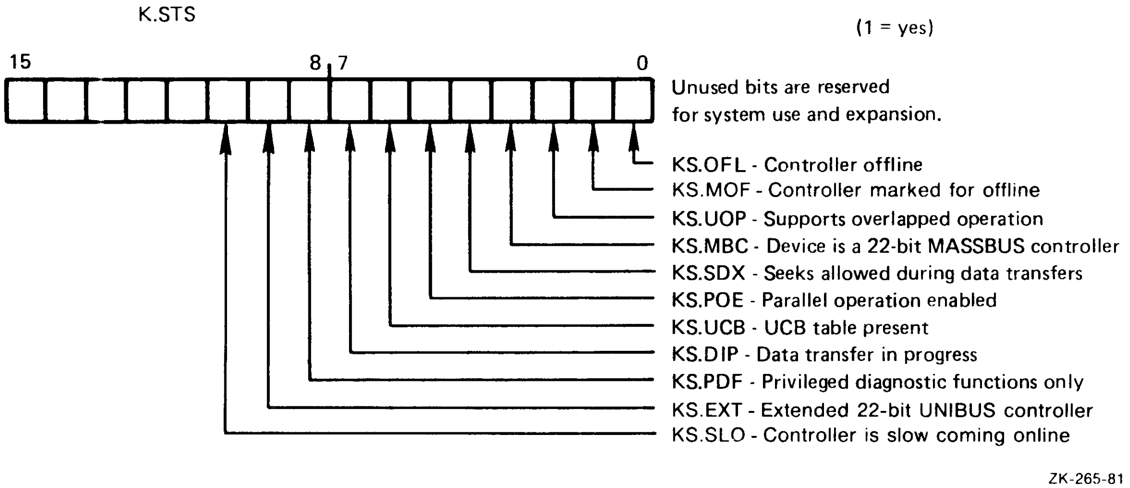

K.STS
(1 = yes)
15
8
7
0
Unused bits are reserved for system use and expansion.
KS.OFL - Controller offline
KS.MOF - Controller marked for offline
KS.UOP - Supports overlapped operation
KS.MBC - Device is a 22-bit MASSBUS controller
KS.SDX - Seeks allowed during data transfers
KS.POE - Parallel operation enabled
KS.UCB - UCB table present
KS.DIP - Data transfer in progress
KS.PDF - Privileged diagnostic functions only
KS.EXT - Extended 22-bit UNIBUS controller
KS.SLO - Controller is slow coming online
ZK-265-81

Figure 4-13: Controller Status Word

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

KS.OFL=1

The Executive reconfiguration routines set this bit to place the controller off-line and clear the bit to place the controller on-line. The bit is used in conjunction with KS.PDF to denote transition states. If a request is made to assign a unit to the controller and this KS.OFL is set (and no other on-line controller is found), the request terminates with the IE.OFL error and a return is made to the driver I/O initiation entry point to get a new packet.

The driver data code should initialize this bit to 1.

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.MBC=10

If this bit is set, the device is a 22-bit MASSBUS controller and does not use UNIBUS mapping registers (UMRs) but has 2 extra registers to describe a 22-bit address. If these registers exist, the offset to the first of them (RHBAE) is in the cell KE.RHB. These registers can be found by using the contents of KE.RHB in conjunction with the contents of S.RCNT. The Executive on-line reconfiguration code calls the common interrupt controller status change routine (in the module DVINT) which dynamically sets or clears this bit during controller processing.

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. Because this bit can be reset by system commands at any time, the driver must not rely on the bit value to decide whether, upon being interrupted, the driver was attempting a seek operation. The driver must 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 and to the Executive common interrupt module DVINT 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.

KS.PDF=400

This bit and one KS.OFL bit indicate the reconfiguration status of the controller. The Executive reconfiguration software accesses both bits to describe the off-line, on-line, and transition status of the controller.

KS.EXT=1000

If this bit is set, the device is a 22-bit UNIBUS controller and does not use UNIBUS mapping registers but has 2 extra registers to describe a 22-bit address. If these registers exist, the offset to the first of them (BAE) is in the cell KE.RHB. These registers can be found by using the contents of KE.RHG in conjunction with the contents of S.RCNT.

KS.SLO=2000

If this bit is set, the controller requires the use of the extended time out feature of the reconfiguration subroutine. If this bit is not set, a controller will transition online/offline immediately.

K.CSR (controller status register address)

Driver access:

Initialized, read-only.

Description:

Contains the address of the Control and Status Register (CSR) for the device controller. Because you can use the CON task to change the CSR value, you need not be overly concerned with initializing K.CSR to the correct value. The driver uses K.CSR to initiate I/O operations and to access, by indexing, other registers that are related to the device and are located in the I/O page. This address need not be the CSR; it need only be a member of the device's register set. The Executive reconfiguration software probes K.CSR to bring a controller on-line. (If probing K.CSR yields a nonexistent memory trap, the controller will not be brought on-line.)

NOTE

> This word is guaranteed to be offset zero for the KRB. This assignment means that an RSX-11M-PLUS driver can access the CSR by the reference @S.KRB and need not use a separate register.

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 UNIBUS mapping register work area extends in a negative direction from the start of the UCB table.

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 (for example, the RK05), 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 is true for RH disks such as the RP06. 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.

K.URM (controller UNIBUS run mask)

Driver access:

Initialized, not referenced.

Description:

This word appears only in a multiprocessor system (that is, M$$PRO is defined).

It contains a UNIBUS run mask that defines the UNIBUS run to which the controller is attached. When controller assignment is made, the cell is moved into S.URM for the fork block there. This word should not be zero.

Table of UCB addresses (offset from K.CSR by K.OFF bytes)

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(R?)+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.

KE.RHB (reserve appropriate amount of storage)

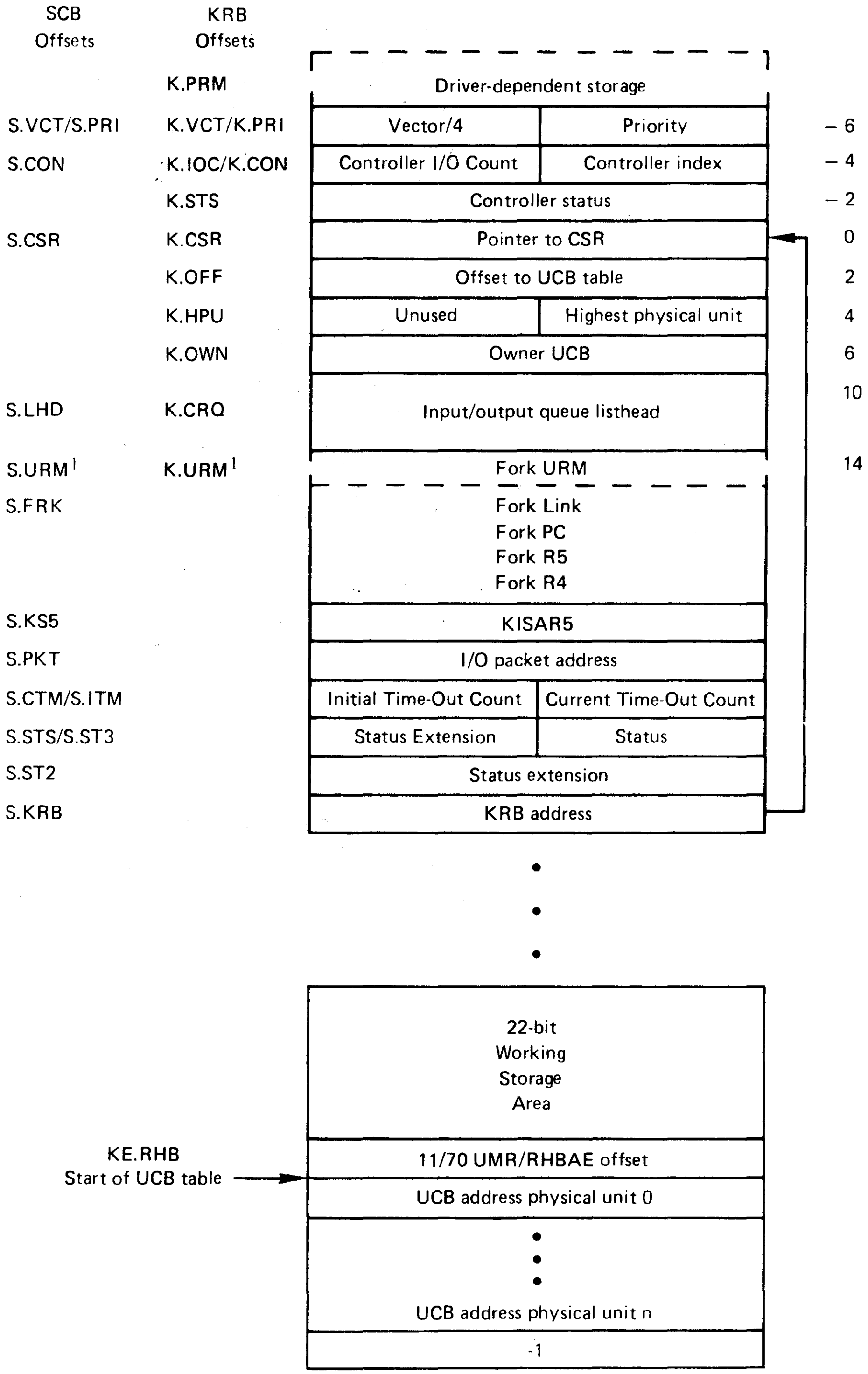
> The UNIBUS mapping register work area extends in a negative direction from the start of the unit control block table. This work area always appears if the device is an NPR device. For devices with either KS.MBC or KS.EXT set, the first word is used as the BAE offset for the controller. This word value is the offset that, when added to the CSR address contained in K.CSR, yields the address of the BAE register on the controller. If both KS.MBC and KS.EXT are clear, the device controller uses UMRs.

4.4.7 Continuous 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 continuous 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. In such a case, although the physical offsets may differ between RSX-11M and RSX-11M-PLUS systems, correct referencing of many locations on both systems is eased. Figure 4-14 shows the physical layout of the continuous KRB and SCB allocation.

4.4.8 Controller Table (CTB)

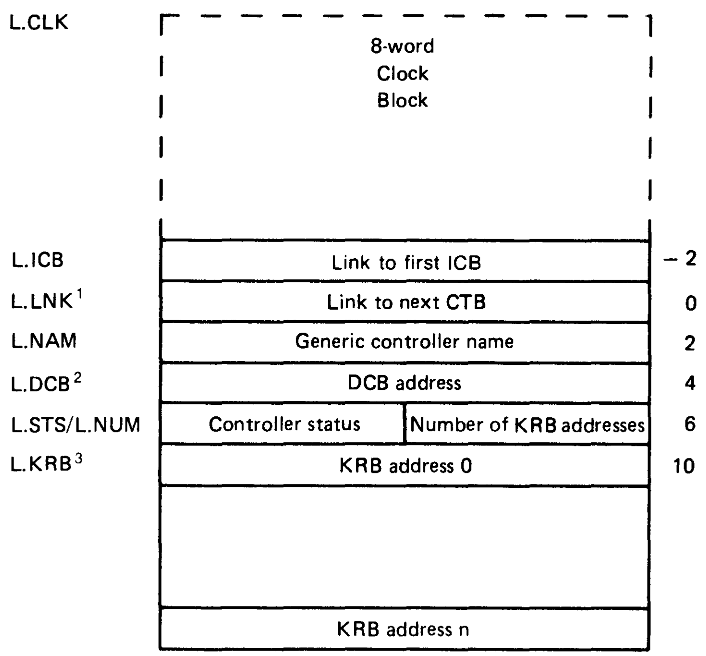
Figure 4-15 is a layout of the controller table. You ensure that the CTB is linked into the system list of controller tables by placing the CTB macro immediately before the allocation of the L.LNK word. The CTB macro generates a global symbol that links the user-written CTB into the system list.



[1] This field is for multiprocessor support (M$$PRO is defined).

ZK-266-81

Figure 4-14: Continuous KRB/SCB Allocation



L.CLK	8-word Clock Block		
L.ICB	Link to first ICB		−2
L.LNK[1]	Link to next CTB		0
L.NAM	Generic controller name		2
L.DCB[2]	DCB address		4
L.STS/L.NUM	Controller status	Number of KRB addresses	6
L.KRB[3]	KRB address 0		10
	KRB address n		

[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.

[3] See Table 4-10 for label XXCTB.

ZK-267-81

Figure 4-15: Controller Table

The fields[1] in the CTB are described below:

L.CLK

Driver access:

Initialized

Description:

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.

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

Description:

This word points to the first interrupt control block for this type of controller. It is a link and not an address. In any system the ICBs must be in an executable pool area. In an I and D space multiprocessor system, they must be distinct for each processor, since each processor has its own local executable pool mapped by KISAR0. Since the linkage must enter and leave other than the usual Executive kernel mapping, the upper 4 bits encode a processor number which may be used to enter $K6TAB, and the lower 12 bits form an address that has been shifted right once. On other than an I and D space multiprocessor system, the upper four bits are considered part of the address, which has still been shifted right once.

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

Driver access:

Not initialized, not referenced.

Description:

All of 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.NAM (2-character ASCII device name)

Driver access:

Initialized, read-only.

Description:

This 2-character ASCII string is the controller mnemonic used to find this controller table from among all the others in the system. For the RH11/70 controller, it is RH instead of DB, DS, DR, or MM.

L.NAM must be unique throughout the system, unlike D.NAM in the device control block.

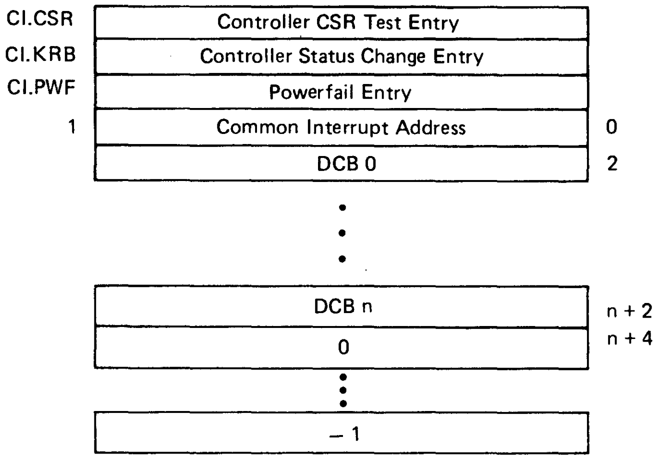
L.DCB (DCB address or address of common interrupt table)

Driver access:

Initialized, not referenced.

Description:

The DCB pointer is used to reach the device control block, and thereby the unit control block and driver dispatch table for a driver. If LS.CIN is set, L.DCB is a pointer to a block that holds the common interrupt address (the address of the interrupt dispatch routine in the Executive), and the DCB addresses (the addresses of the DCBs for the devices that this controller interfaces). This block is called the common interrupt table and is shown in Figure 4-16.



[1] If LS.CIN is set, L.DCB in CTB points to this structure instead of to the DCB.

ZK-268-81

Figure 4-16: Common Interrupt Table and Table of DCB Addresses

The powerfail entry at offset CI.PWF and the controller status change entry at offset CI.KRB are addresses of routines built into the Executive and are used instead of the entries in a particular driver dispatch table. This allows devices that have no DCB (for example, the interprocessor interrupt and sanity timer) to still participate in reconfiguration.

At offset CI.KRB is the address of a routine built into the Executive for multidriver controllers such as the RH type. This routine should set or clear the KS.MBC bit to indicate whether the device is connected to an RH11 or an RH70. The driver checks the KS.MBC bit to determine which addressing format to use. If the value at CI.CSR is zero, the Executive on-line routines check the existence of a device attached to this controller by probing the address at K.CSR. If the value is nonzero, it is the address of a routine built into the Executive to check device presence. Instead of probing the address at K.CSR, the Executive on-line code calls this routine, which returns either with the C bit clear if the device is present or with the C bit set if the device is not present.

The common interrupt table may have only the common interrupt address in those cases in which a DCB does not exist (for example, the IIST). If LS.MDC is clear, then only one DCB address exists. (The zero termination is still necessary.) If LS.MDC is set, then more than one DCB address is possible; therefore, space should be left for all possible DCB addresses (for LOAD) and the table terminated by a zero, followed by a -1. Empty entries in this case are indicated by a zero word. LOAD will then enter the DCB addresses into the table when it loads data structures for drivers.

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.

The maximum value for L.NUM depends on the type of device and on whether the driver is loadable. For common interrupt devices, the value must be less than 17 (decimal). For resident drivers and drivers loaded by MCR LOAD, the value must be less than 17 (decimal). For drivers loaded by VMR LOAD, the value must be less than 17 (decimal) if the data base is loadable and less than 129 (decimal) if the data base is resident.

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.

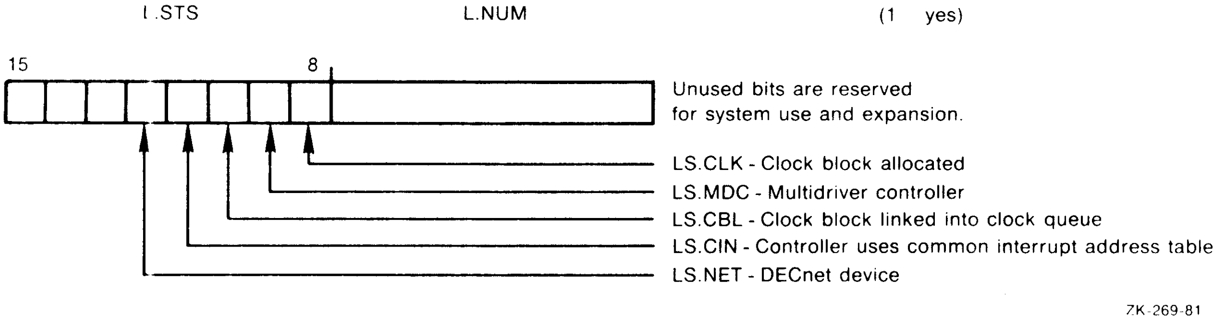

L.STS
L.NUM
(1 yes)
15
8
Unused bits are reserved for system use and expansion.
LS.CLK - Clock block allocated
LS.MDC - Multidriver controller
LS.CBL - Clock block linked into clock queue
LS.CIN - Controller uses common interrupt address table
LS.NET - DECnet device
ZK-269-81

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.

LS.CIN=10

If this bit is set, the driver is associated with a common interrupt controller and must have exactly one interrupt vector. The driver is therefore called at the D.VPWF entry point only for unit power failure. The Executive uses the CI.PWF entry point in the common interrupt entry table for controller power failure recovery. In addition, the cell L.DCB does not point to the device control block but rather to the common interrupt entry table in the Executive.

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 system-wide 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(R?)) 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, where xx are the characters comprising the controller mnemonic. Because LOAD supplies this address in the DDT when it loads the driver, a loadable driver should not specify this address in the DDT.

NOTE

A KRB address of zero indicates a controller that was specified during system generation with no attached units. No controller request block for such a controller is generated.

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 LOAD (both VMR and MCR).

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.

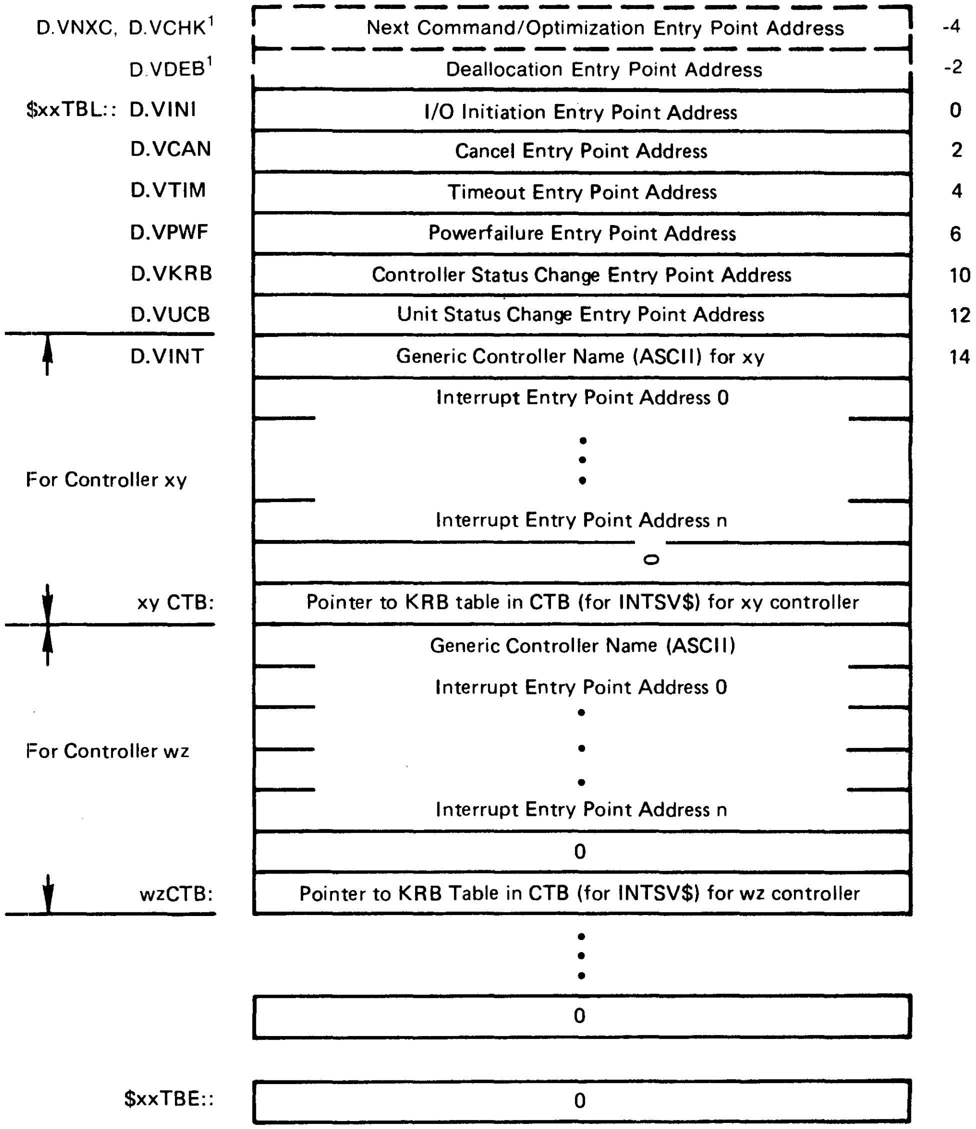
NOTE

> 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
$xxTBL::	Defines the start of the DDT. You specify this label in the D.DSP word of the DCB of resident drivers to link the DCB to the DDT. For loadable drivers, the LOAD routines use this label to fill in D.DSP.
xxCTB:	Defines the pointer to the table of KRB addresses in the CTB of the controller for device xx. Because a driver can support different types of controllers, there may be more than one of this form of label. (The DDT$ macro supports only one controller type.)
$xxTBE::	Defines the end of the DDT for Executive LOAD and UNLOAD 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
$xxTBL:: D.VINI	I/O Initiation Entry Point Address	0
D.VCAN	Cancel Entry Point Address	2
D.VTIM	Timeout Entry Point Address	4
D.VPWF	Powerfailure Entry Point Address	6
D.VKRB	Controller Status Change Entry Point Address	10
D.VUCB	Unit Status Change Entry Point Address	12
D.VINT	Generic Controller Name (ASCII) for xy	14
For Controller xy	Interrupt Entry Point Address 0	
	• • •	
	Interrupt Entry Point Address n	
	0	
xy CTB:	Pointer to KRB table in CTB (for INTSV$) for xy controller	
For Controller wz	Generic Controller Name (ASCII)	
	Interrupt Entry Point Address 0	
	• • •	
	Interrupt Entry Point Address n	
	0	
wzCTB:	Pointer to KRB Table in CTB (for INTSV$) for wz controller	
	• • •	
	0	
$xxTBE::	0	

[1] These are optional advance driver features

ZK-270-81

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:

```
$xxTBL::
xxCTB:
$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[1]	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

[1] The characters xx are the 2-character mnemonic.

The Executive reconfiguration software uses the following rules when it accesses the interrupt address block to calculate the vectors for a controller. To calculate the first vector address, reconfiguration routines access the cell K.VCT (or S.VCT) in the controller request block. If K.VCT is not equal to zero, it is multiplied by 4. The result is the vector address that will be loaded with the address found in interrupt entry point 0. The next interrupt entry point is examined. If it is zero, there are no more vectors or interrupt entry points for the controller. If it is even, the next vector address is the previous one plus 4 and that vector address is loaded with the entry point address just examined.

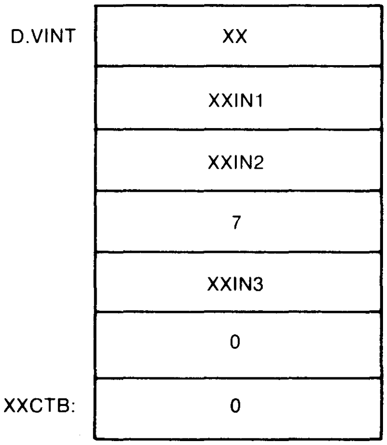
If an entry point value in the block is odd (bit zero is set), bit zero is cleared and the resulting number is an offset to the next vector address. To compute the next vector address, the offset is added to the last vector address The next interrupt entry point is examined. If it is even, then its value is loaded into the last vector address computed. If it is odd, the result is an offset that is added to the vector address just computed and the next entry point is examined. The computation of vector addresses terminates when the next entry point is zero.

The entries shown in Figure 4-19 can be used to calculate the interrupt vector addresses when K.VCT equals 300.

The vectors at 300 and 304 are loaded with addresses xxIN1 and xxIN2. The odd value 7 yields the offset 6 that is added to the last vector computed to attain 312. The address xxIN3 in the next interrupt entry point examined is loaded in the vector at 312. A zero word in the block shows there are no more vectors or interrupt entry points.

Following the interrupt entry address block for a controller type is a pointer to the KRB table in the CTB. Its label is in the form xxCTB, which is used by the INTSV$ macro. This pointer connects the driver code to the driver data base and is the last entry in a block for a specific controller.

A zero terminates the driver dispatch table. The global label in the form $xxTBE marks the terminating word in the DDT.



ZK-271-81

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 DRQIO. 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
```

You may find more information on and coding requirements for the queue-to-driver operation in the description of the UC.QUE bit in Section 4.4.4 and an example of its use in Chapter 8.

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 continuous 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.

On a multiprocessor system, before returning a packet to the driver, $GTPKT calls the conditional fork routine $CFORK to ensure that the driver executes on the correct processor. If the current processor is the correct processor, $CFORK returns to $GTPKT, and $GTPKT dequeues an I/O packet, queues it to the driver, and returns to the driver with the C bit clear. Should the current processor not be the correct processor, $CFORK will call $FORK which returns to the driver with the C bit set. This action causes the driver to dismiss itself. Eventually the fork processor restarts the driver executing on the correct processor.

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 Next Command Entry Point

Ths offset D.VNXC in the driver dispatch table is only applicable to the terminal driver. The offset D.VNXC contains the entry point address of a routine within the terminal driver which is called from the routine $SNCMD in the Executive module DRSUB. This entry point is entered when a task exits whose TI: is set to serial mode. The driver then passes the next CLI command to the MCR dispatcher. When the Executive enters the driver, the following register conventions pertain:

```
R0 = UCB address of the TI: of the exiting task.
```

4.5.6 Queue Optimization Entry Point

The offset D.VCHK in the driver dispatch table contains the address of this entry point. The routine $DRQRQ in the Executive's module DRSUB calls the driver at this entry point at priority zero. When the Executive enters the driver, the following register conventions pertain:

```
R5 = UCB address
R1 = I/O packet address
```

If the I/O operation is a data transfer function, the I/O packet contains the starting LBN for the I/O request. The routine at this entry point must verify the request is a data transfer function, and if it is, the routine must replace the starting LBN with the starting cylinder, track, and sector number to perform queue optimization. See the routine DBCHK in the module DBDRV for an example of a driver that supports queue optimization.

4.5.7 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.

4.5.8 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.

If the driver supports a common interrupt device (that is, the LS.CIN bit in the CTB is set), the driver is called at this entry point only for unit power failures. For controller power failures, the Executive calls the entry point at CI.PWF in the common interrupt entry table. See the description of the offset L.DCB in Section 4.4.8.

4.5.9 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.

> NOTE
>
> If the controller is a common interrupt controller (LS.CIN is set), the Executive does not call the driver at this address (if any) specified in the DDT but at the address in the common interrupt table labelled CI.KRB. See Section 4.4.8.

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. (The I/O error code symbols are listed in an appendix of the IAS/RSX-11 I/O Operations Reference Manual.)

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. However, the terminal driver (TTDRV) rejects a controller on-line request for a DZ11 multiplexer if some of the status bits indicate that the device is not a DZ11 or that it is broken.

4.5.10 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. (The I/O error code symbols are listed in an appendix of the <u>IAS/RSX-11 I/O Operations Reference Manual</u>.)

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.

4.5.11 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.

If the driver is loadable, it 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.

If the driver is resident, it receives control directly from the interrupt vector. It runs at priority PR7 and the low-order four bits of the PS have the controller number of the interrupting device. Because the low-order four bits are status bits and almost any instruction modifies them, the first operation that should be performed is to save the PS. Then, the driver does its processing at priority PR7 (saving registers if necessary). After processing, it restores the registers (if necessary) and dismisses the interrupt by executing an RTI instruction.

However, all reasonable drivers should use the INTSV$ macro call at an interrupt entry point. The INTSV$ macro resolves entry processing for both loadable and resident drivers. For loadable drivers, INTSV$ does not generate a call to $INTSV because LOAD 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. (LOAD previously set the priority in the interrupt control block based on the value at offset K.PRI in the controller request block.)

For resident drivers, INTSV$ generates a call to the $INTSV coroutine, which sets the priority to that specified in the INTSV$ macro call; saves registers R4 and R5; and forms the controller index from the PS and stores it in R4.

For both loadable and resident drivers, 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 pertain for both loadable and resident drivers:

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

In summary, then, the INTSV$ macro eliminates your having to consider the coding differences between a loadable and a resident driver in the interrupt service routine.

4.5.12 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:

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 RSX-11M-PLUS

This chapter describes how to incorporate a user-supplied driver into an RSX-11M-PLUS system. The material in the chapter depends on your having created source code according to the programming specifics given in Chapter 4.

5.1 GUIDELINES FOR INCORPORATING A DRIVER

The procedures that you follow to incorporate a user-supplied driver into RSX-11M-PLUS depend on the type of driver you have. Your driver is one of the following types:

- Loadable driver with a loadable data base
- Loadable driver with a resident data base
- Resident driver with a resident data base

If your driver is loadable with a loadable data base, you may perform a system generation to include your driver, or you may incorporate it directly into your currently running system. If you want to use a new version of a loadable driver with a loadable base, and your driver is currently loaded, you must create a new system image file, load the new version of the driver into the file, and then bootstrap the new system. Refer to Section 5.1.1 if you want to incorporate your driver at system generation. Refer to Section 5.1.2 if you want to incorporate your driver after system generation.

If your driver is loadable with a resident data base, or is resident, you must perform a system generation because the resident driver and/or data base reside in the Executive and must be assembled and task built as part of the Executive. Refer to Section 5.1.1 to incorporate your driver at system generation.

Because loadable drivers and loadable data bases can be changed and reloaded without performing a system generation, loadable drivers with loadable data bases are easier to debug and maintain than resident drivers and/or resident data bases.

5.1.1 Incorporating a Driver at System Generation

If you want to build a loadable driver with a loadable or resident data base during system generation, proceed as follows:

1. Assemble and task build your driver to eliminate any assembly or Task Builder errors.

2. Put the MACRO-11 source files containing your driver code and data base in UFD [11,10] on the target system disk. The driver source file should be named xxDRV.MAC and the data base source file should be named xxTAB.MAC, where xx is the 2-character device mnemonic. Mnemonics for user-supplied devices should begin with the letters J or Q to avoid conflict with DIGITAL-supplied devices.

3. Perform a system generation and choose the Full-functionality Executive. Answer the questions concerning user-supplied drivers printed during system generation. This procedure includes your driver data base in the Executive if it is resident and builds your driver task image. A loadable driver and data base are loaded into the system image file. Refer to Section 5.3 for a description of the system generation procedure.

4. Use CON from MCR to make your devices accessible. Refer to Section 5.2.5 for the CON command description.

If you want to build a resident driver, proceed as follows:

1. If your driver can run loadable with a loadable data base, first build and test it as loadable with a loadable data base.

2. Put the MACRO-11 source files containing your driver code and data base in UFD [11,10] on your target system disk. The driver source file should be named xxDRV.MAC and the data base source file should be named xxTAB.MAC where xx is the 2-character device mnemonic. Mnemonics for user-supplied devices should begin with the letters J or Q to avoid conflict with DIGITAL-supplied devices.

3. Perform a system generation. If you want to include a resident driver, you must not choose the Full-functionality Executive or Executive data space support. Answer the questions concerning user-supplied drivers printed during system generation. This procedure includes your driver and data base modules in the Executive. Refer to Section 5.3 for a description of the system generation procedure.

4. Use CON from MCR to make your devices accessible. Refer to Section 5.2.5 for the CON command description.

5.1.2 Incorporating a Loadable Driver with a Loadable Data Base After System Generation

The procedures to incorporate a loadable driver with a loadable data base after system generation involve the following steps:

1. Assemble and task build your driver to eliminate any assembly or Task Builder errors.

2. Put the MACRO-11 source files containing your driver code and data base in UFD [11,10] on your target system disk. The driver source file should be named xxDRV.MAC and the data base source file should be named xxTAB.MAC, where xx is the 2-character device mnemonic. Mnemonics for user-supplied devices should start with the letters J or Q to avoid conflict with DIGITAL-supplied devices.

3. Run the system generation procedure and follow the instructions in the "Adding a Device" section. (For information on invoking the system generation procedure (SYSGEN), refer to the RSX-11M-PLUS System Generation and Installation Guide).

   The system generation procedure asks you to enter the 2-character device mnemonic for your driver. Remember, this should be the same mnemonic used in the driver and data base source file names.

4. Use the MCR LOA command to link your driver data base into the system device tables and to load your driver data base and driver code. Refer to Section 5.2.4 for the LOA command descriptions.

5. Use CON from MCR to place the controller(s) and unit(s) on line. (CON can also alter vector assignments.) Refer to Section 5.2.5 for the CON command descriptions.

5.2 WHAT THE SYSTEM GENERATION PROCEDURE DOES FOR YOU

The system generation procedure assembles your driver and data base, puts the resulting object modules in the Executive object library and task builds your driver. If your driver or its data base is resident, the driver and/or data base is included in the Executive. If your driver or its data base is loadable, the driver and/or data base is loaded into the system image file. You must then make the controller(s) and unit(s) accessible.

The commands that the system generation procedure uses to assemble your driver and data base, insert your driver and data base modules in the library, and task build your driver, are the same commands that you may use to assemble, insert and task build your driver. The following subsections explain each of the procedures for incorporating your driver.

5.2.1 Assembling the Driver and Data Base

The system generation procedure assembles your driver and its data base with the following commands:

```
MAC>[11,24]xxDRV,[11,34]xxDRV/-SP=[1,1]EXEMC/ML,[11,10]RSXMC/PA:1,xxDRV
MAC>[11,24]xxTAB,[11,34]xxTAB/-SP=[1,1]EXEMC/ML,[11,10]RSXMC/PA:1,xxTAB
```

If your driver is resident, these commands are located in the file RSXASM.CMD. If your driver is loadable, these commands are located in the file xxDRVASM.CMD, where xx is the device mnemonic.

The commands to the assembler specify as input the Executive macro library EXEMC.MLB, the Executive assembly prefix file RSXMC.MAC, and either your driver code or driver data base source file (xxDRV.MAC or xxTAB.MAC). EXEMC.MLB contains the macro definitions of structures and symbolic offsets that your code may reference. (The source code for some of the macro definitions is given in Appendix A.) RSXMC.MAC contains symbols defined during system generation and definitions of some macros that your driver may invoke (such as DDT$, GTPKT$, and INTSV$). The assembler looks for the source file of your driver in UFD [11,10].

As output, the assembler creates object modules in UFD [11,24] and listing files in UFD [11,34]. The object modules xxDRV.OBJ and xxTAB.OBJ will later be put in the Executive object library. You should retain the listing files xxDRV.LST and xxTAB.LST for documentation and maintenance purposes.

5.2.2 Inserting the Driver and Data Base Modules in the Library

After your driver and data base modules have been assembled, the driver and data base modules are added to the Executive object library. Commands to the Task Builder (described in Section 5.2.3) require the modules be in this library.

The system generation procedure uses the following commands to add both the driver and its data base to the same library:

```
     LBR [1,24]RSX11M/RP=[11,24]xxDRV,xxTAB
```

The command to LBR adds the object modules of both your driver and its data base to the Executive object library RSX11M.OLB, which resides in UFD [1,24]. RSX11M.OLB is built from object modules assembled during system generation. The /RP switch ensures that any modules of the same name are replaced by the recently created modules. If this is not the first time you have performed this operation, LBR prints messages telling you that it replaced your modules in the library with the new versions.

5.2.3 Task Building the Driver

After the modules have been added to the Executive object library, the system generation procedure task builds your driver and data base. The commands for a resident driver are located in the file RSX11M.CMD. The commands for a resident data base are located in the file RSX11M.CMD on systems without Executive data space support and in the file DSP11M.CMD on systems with Executive data space support.

The commands for a loadable driver are located in xxDRVBLD.CMD where xx is the device mnemonic. The following discussion explains each of the lines that are contained in the command file for a loadable driver:

1. When the system generation procedure builds your driver, a task-image file name and a symbol definition file name are specified as TKB output. The task image and symbol definition files are placed in the UFD corresponding to the system UIC that will be in effect when the LOA command is issued. The file names are both xxDRV, where xx is the device mnemonic. The Task Builder produces the output files named xxDRV.TSK, xxDRV.MAP, and xxDRV.STB. For example, the input supplied to TKB to build the xx device would look like the following:

   ```
   [1,54]xxDRV/-HD/-MM,[1,34]xxDRV/-SP,[1,54]xxDRV=
   ```

2. No task header is included. The switch /-HD is used, as in the previous example. A driver is not really a task, but an extension of the Executive, and as such needs no task header.

3. The switch /-MM must be used in the command line.

4. A map file is produced and is useful for debugging. All driver map files are written to UFD [1,34]. The switch /-SP suppresses automatic spooling to the line printer.

5. The system generation procedure links your driver to the system symbol definition file that contains definitions of Executive global symbols. Continuing the example from item 1 above might give further TKB input that would like this:

   ```
   [1,24]RSX11M/LB:xxDRV:xxTAB
   [1,54]RSX11M.STB/SS
   ```

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

   The second line in item 5, above, indicates that the symbol definition file RSX11M.STB is to be searched selectively (/SS) for definitions of Executive global symbols. Note that the /SS switch must appear in this context. It is never omitted.

6. The system generation procedure links your driver to the system library file that defines masks and offsets used in the Executive. To continue the example:

   ```
   [1,1]EXELIB/LB
   /
   ```

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

7. The Task Builder is directed not to allocate space for a stack within the driver.

   ```
   STACK=0
   ```

8. A partition for the driver is specified:

   ```
   PAR=DRVPAR:120000:40000
   //
   ```

   The partition name DRVPAR is the typical name of a conventional partition reserved for drivers. A driver may be loaded into any system-controlled partition. The base virtual address of the partition is always 120000 (8). That is, the loadable driver must be mapped through kernel APR5. The length of the partition, the second parameter should not exceed 8K words (40000 octal bytes).

   The double slash ends the option phase of the Task Builder.

5.2.4 Loading the Driver

After your driver is task built, you are ready to load the driver on your system. This procedure is used when you are incorporating your loadable driver with a loadable data base after system generation. Loading is done by using the privileged MCR command LOAD. Its form is:

```
>LOAD xx:[/PAR=GEN][/HIGH]
>
```

The variable xx is the 2-character device mnemonic. Specifying a partition is optional. If a partition is not specified, the partition input to the Task Builder is used. The keyword /HIGH puts the driver as high as possible in the partition. The default condition is to put the driver as low as possible in the partition.

LOAD performs many diagnostic checks on your driver data base, relocates many addresses within the data base, and loads the data base and the driver code into memory. Because the LOAD diagnostic checks are complicated and LOAD supports another, infrequently used option (/CTB), a description of LOAD is given in Section 5.4. LOAD error messages and meanings are listed in the RSX-11M/M-PLUS MCR Operations Manual. After the driver is loaded, the controller(s) and units are off-line and are not accessible. To allow access to the device, you must next place the controller(s) and unit(s) on-line.

5.2.5 Making the Devices Accessible

After your driver has been successfully loaded, you must make the controller(s) and units accessible. You use the CON task to place controller(s) and units on-line, to change vector and CSR assignments that you established in the driver data base, and to take units and controller(s) off-line. Unless the vector and CSR values in the driver data base are not correct for the running system, you can place the controller(s) and units on-line. You may change the vector and CSR assignments to match the hardware CSR and vector assignments only while the controller(s) and units are off-line.

5.2.5.1 Setting Vector and CSR Assignments - If the values at the offsets S.VCT/K.VCT and S.CSR/K.CSR in the KRB(s) of your driver data base are incorrect for the running system, you must issue the privileged SET command in CON to establish the correct values.

NOTE

> Because CON causes the Executive to access a driver data base when it changes a vector or CSR assignment, you must load the driver before you issue the SET commands in CON. If a driver data base is resident, you do not need to load the driver to establish correct vector and CSR assignments.

You must do this operation while the controller(s) and units are off-line. (LOAD ensures that, for a loadable data base, the controller(s) and unit(s) are off-line.) Typical commands to set a vector and CSR for a driver that supports a single controller are as follows:

```
>CON
CON>SET xxA VEC=300 CSR=160040
CON>^Z
>
```

The command first establishes 300 as the vector for controller A of type xx. The Executive accesses the offset S.VCT/K.VCT in the driver data base and writes the specified value divided by 4. The command secondly establishes the control and status register address as 160040 for controller A of type xx. The Executive accesses the offset S.CSR/K.CSR in the driver data base and writes the specified value. You type the CTRL/Z combination to exit from CON. After you set the vector or CSR assignment, you can attempt to place the controller(s) and units on-line.

5.2.5.2 Placing a Controller and Units(s) On-Line

- If the vector and CSR assignments in your driver data base are correct for the running system, you can place the controller(s) and units on-line by issuing the privileged ONLINE command in CON.

NOTE

> Because placing a controller or a unit on-line causes the Executive to call the driver, you must have loaded the driver before you issue the ONLINE command in CON.

The following commands demonstrate a typical sequence to place a single controller and two attached units on-line.

```
>CON
CON>ONLINE xxA
CON>ONLINE xx0:
CON>ONLINE xx1:
CON>^Z
>
```

The first command places controller A of type xx on-line. The Executive accesses the KRB of the controller to read the S.VCT/K.VCT offset and initializes the vector to point to the related interrupt control block.

NOTE

> If the driver is resident within the Executive, the vector points directly to the driver. For a common interrupt controller, the vector points to the interrupt entry address in the Executive rather than to an ICB.

The Executive then ensures that the address in S.CSR/K.CSR is valid (that is, some device responds at that address). Refer to Section 5.2.5.3 for a discussion of CSR and vector assignment errors. Next, the Executive calls the driver at its controller status change entry point. Only after the driver indicates success does the Executive change the status bit in the SCB/KRB of the controller from off-line to on-line.

The second and third commands place logical units 0 and 1 on-line. The Executive checks that the controller is on-line (that is, an access path exists to the unit). If the controller is not on-line, the Executive sets the UCB of the unit as marked for on-line. (The Executive automatically places on-line a unit that is in the marked for on-line state only when its controller is placed on-line.) If the controller is on-line, the Executive calls the driver at its unit status change entry point. Only after the driver indicates success does the Executive change the status bits in the UCB of the unit from off-line to on-line. (The driver is not required to take any special action to indicate success. Refer to the discussion of status change entry points in Section 4.5.)

After you have issued the ONLINE commands, you can issue the DISPLAY command in CON as follows to verify that the devices are in the state that you want them to be in:

```
>CON
CON>DISPLAY FULL FOR xx

                    (The display appears at the terminal.)

CON>^Z
>
```

The command displays status of all controllers of type xx and of all units attached to the controllers.

5.2.5.3 **CSR and Vector Assigment Errors** - CSR and vector assignment errors are not always immediately detectable. When you issue the ONLINE command to CON to place a device on-line, CON verifies that some device responds at the CSR address that you established at the S.CSR/K.CSR offset in your driver data base. CON can encounter one of three possible cases:

- Your device is at the established CSR address and it responds to the CON probe. This is the case you want. CON continues attempting to place your device on-line.
- Your device is at some address in the I/O page other than the established CSR address, but some other device responds at the established CSR address. CON cannot distinguish your device from some other device, and continues attempting to place your device on-line possibly resulting in a system hang or crash.
- Your device is at some address in the I/O page other than the established CSR address, and no device responds at the established CSR address. In this case, CON reports an error and does not place the device on-line.

Therefore, if CON places your device on-line and the device subsequently does not respond, have a DIGITAL Field Service representative verify the CSR address jumpers and ensure that the CSR assignment in your driver data base matches the verified hardware CSR assignment.

When the vector address developed from the value that you established at the offset S.VCT/K.VCT in the driver data base differs from the hardware vector assignment, several outcomes are possible. Should the established vector already be in use (that is, pointing at other than the nonsense interrupt entry address in the Executive), CON reports the condition and does not place the device on-line. If the vector is not in use, CON establishes it as the device vector and continues attempting to place the device on-line. This action does not guarantee that the software and hardware vector assignments match.

When CON does place a device on-line and the software and hardware vector assignments do not match, two results are possible:

- Your driver will time out waiting for an interrupt.
- The device will interrupt through an unused vector.

If error logging is active on your system, a nonsense interrupt will be logged as an undefined interrupt error and the ERRSEQ count in the Executive is increased by 1. The RMDEMO task display, which includes the ERRSEQ count, will reflect the occurrence of nonsense interrupts by an increasing number in ERRSEQ. Consult an error log report and look for undefined interrupt errors.

When (1) error logging is active, (2) nonsense interrupts do not occur, and (3) your driver times out, the interrupt could be going through some other driver vector. If the unexpected interrupt goes to a DIGITAL-supplied driver, one of two outcomes is possible.

- The interrupt will simply be dismissed. (Common interrupt routines dismiss unexpected interrupts and some drivers keep track of when they expect interrupts and dismiss unexpected ones.)
- The driver will react in an unpredictable fashion (such as attempting to terminate the last I/O packet again) causing a system crash.

Thus, error logging and the ERRSEQ count in the RMDEMO display help indicate improper vector assignments.

5.3 USER-SUPPLIED DRIVER SYSTEM GENERATION DIALOGUE SUMMARY

If you are building either a loadable driver with a resident data base or a resident driver, you must perform a system generation to incorporate your driver into the system. This section summarizes the system generation dialogue only as it relates to user-supplied driver support and related features. For more information on the system generation procedure itself, refer to the <u>RSX-11M-PLUS System Generation and Installation Guide</u>.

NOTE

> If you are building a loadable driver with a loadable data base, you need not perform a system generation to incorporate your driver. However, you can still build your driver during system generation. Section 5.1.2 describes the complete procedures to build a loadable driver with a loadable data base any time after you build the Executive under which the driver will run.

5.3.1 Choosing Executive Options

The system features are determined during the "Choosing Executive Options" section. You have to specify answers related to including a user-supplied driver in your system. A question in the following form appears:

```
> Do you want the Full-functionality Executive?  [Y/N D:Y]:
```

If you choose the Full-functionality Executive, your driver must be loadable with either a loadable or resident data base. If you want to incorporate a user-supplied resident driver, you must omit the Full-functionality Executive and omit Executive data space support. All DIGITAL-supplied drivers should be loadable with a loadable or resident data base.

If you do not choose the Full-functionality Executive, the system generation procedure asks the two following questions:

```
> Do you want Executive data space support?  [Y/N D:N]:
```

If you have a loadable driver with either a loadable or resident data base, you should answer Yes to this question. If you have a resident driver, you must answer No to this question.

```
> What is the ICB pool size (in words)?  [D R:16.-1024.  D:128.]:
```

On systems with Executive data space, the ICB pool must be large enough for all the drivers loaded into the virgin system image. One ICB (8 words) is needed for every 16(10) controllers of the same type. If the device controlled by your driver has a large number of controllers, you should ensure that there is enough ICB pool space.

Whether you choose the Full-functionality Executive or not, another question in the following form asks about XDT support:

```
> Do you want to include XDT?  [Y/N D:N]:
```

You should answer Yes to this question. XDT (described in Chapter 6) is helpful in debugging system problems which incorporating a faulty driver may engender.

After this question, there are no more questions in this section concerning user-supplied driver support or related features.

5.3.2 Choosing Peripheral Configuration

In the Peripheral Configuration section of the system generation procedure, you must answer questions about your driver and its data base configuration. A question in the following form asks you to supply your device mnemonics:

```
> Enter device mnemonics for user-supplied drivers [S]:
```

You must enter the 2-character device mnemonic for your driver. This should be the same mnemonic used in the driver and data base source file names.

If you did not select Executive data space support, a question in the following form appears:

```
> Do you want the xx:  driver to be loadable?  [Y/N D:N]:
```

Answer Yes to this question if you want a loadable driver. Answer No if you want your driver to be resident.

If your driver is loadable, the next question in the system generation procedure asks you about your data base:

```
     > Do you want the xx:  driver's data base to be loadable?  [Y/N D:N]:
```

Answer Yes to this question if you want a loadable data base. Answer No if you want your data base to be resident.

The system generation procedure always asks you to specify the highest interrupt vector address:

```
     > What is the highest interrupt vector address?  [O R:n-774 D:n]:
```

The system generation procedure calculates and displays the highest interrupt vector address needed for the DIGITAL-supplied devices. If the vector address for your device is higher than this, enter the highest vector address used by your device.

This ends the system generation portion of incorporating a user-supplied driver. If you are generating a new system, the system generation procedure includes your driver in the Executive if it is resident, or loads your driver into the system image if it is loadable. After the newly built system is running, you must make the devices that your driver supports accessible, as directed in Section 5.2.5.

If you are adding your driver after system generation, you must load your driver and make the devices that it supports accessible, as described in Sections 5.2.4 and 5.2.5.

5.4 LOAD PROCESSING

The Executive LOAD routines extensively check the driver data base; LOAD provides the /CTB switch to handle multidriver controllers. The following subsections describe the two aspects of LOAD.

5.4.1 LOAD Operations and Diagnostic Checks

Two modules (LDVLDB and LDVFIN) in LOAD, load a driver into memory: one conditionally checks the validity of and loads the data base; and the other finishes the operation by loading the driver. If there is no resident data base, the data base is loaded into the system pool. The LOAD routines relocate and validate many of the pointers within the data base and, in the process, validate other data in the structures. (If the data base is resident, no validity checks on the driver data base are performed.) The driver itself is then loaded into its partition, and the interrupt control blocks are created.

To read the data base from the xxDRV.TSK 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 LOAD routines must know the starting address of the DCB. If the global label $xxDCB is not defined (that is, not in the symbol table file), the start of the DCB is assumed to be the first word of the data base. Many unusual error conditions result when LOAD assumes that the DCB is at the start of the data base and the DCB is elsewhere in the data base and not labelled properly. Thus, 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. If L.DCB points to a common interrupt table, the common interrupt entry point address in the table must be even and lie within the Executive. 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.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 resident or 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
K.CRQ+2 — The listhead for the controller request queue. It is initialized to an empty list with the first word zero, and the second word pointing to the first, relocated.

The following offset in the KRB is checked:

K.URM — In a multiprocessor system, the UNIBUS run mask for the controller must have exactly one bit set and that bit must correspond to an existing UNIBUS run (either primary or secondary).

LOAD puts each controller in the off-line state by setting the KS.OFL bit in the K.STS byte. Therefore, all controllers are off-line until you use CON to place each one on-line.

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.

The following offsets in the DCB are checked:

D.NAM — The device name must be the same as that which you specified in the LOAD command line.

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 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.

LOAD places each unit in the off-line state by setting the US.OFL bit in the U.ST2 byte. Therefore, all units are off-line until you use CON to place each one on-line.

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. (LOAD 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 vector address allowed on the system (as specified at system generation) or does not point to a nonsense interrupt entry point, LOAD prints a warning message. You can use CON to set the correct vector address before you place the controller on-line. 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 4.5.1. If the device that the data base describes does not have any physical controllers (that is, the value at offset S.VCT/K.VCT equals zero), the DDT is not checked. If S.VCT/K.VCT is nonzero, 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, LOAD 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 global label $xxCTB, is loaded into the DDT.

5.4.2 Use of /CTB in LOAD

Some controllers such as the RH70 can support more than one device type. The CTB for such a controller differs in two ways from the standard CTB. First, the table of KRB addresses at the end of the CTB contains pointers to KRBs of controllers for different device types. Second, instead of a pointer to one DCB in the CTB, there is a pointer to a table of DCB addresses because each different device must have a separate DCB to describe each separate device type. Morever, more than one driver supports the different types of devices capable of being attached to the controller.

The data base for such a controller must necessarily be split. Because only one CTB is needed to describe the type of controller, only one driver that supports a device on that controller type can define that CTB. The remaining drivers cannot define a CTB but must reference the CTB defined for the first driver. Because all drivers and their data bases can be loadable, the remaining drivers and the syntax in the LOAD command must indicate to the LOAD routines which resident CTB to use. (Of course, the driver data base that defines the CTB of the multidriver controller must be loaded or already resident before the other drivers can be loaded.)

The driver data base that defines the CTB for a multidriver controller allows for structures to define the data base of drivers that are not resident. In particular, for each device controller there must be a slot in the CTB table of KRB addresses to hold the pointer to the KRB. (A KRB must be defined to describe each occurrence of a controller.) A zero is in the pointer for a device whose data base (and therefore, whose KRB) is not resident. Moreover, the table of DCB addresses in the common interrupt table must have sufficient slots to point to the DCBs of all device types that the controller supports. A zero in the DCB table indicates no DCB exists (that is, the data base for a device type is not resident).

To load the data base of a device attached to the multidriver controller, the LOAD routines must know the controller name, the location of the device on the MASSBUS controller and the KRB(s) of the device(s) whose driver is to be loaded. The /CTB syntax in the LOAD command supplies the first two pieces of information. For example:

```
>LOA DR:/CTB=RHB,C
>
```

The letters RH are the name in the CTB already resident in the system. LOAD routines search the system list of CTBs to locate the correct one. The letters B and C are the slots in the table of KRB addresses that will be used to link the resident CTB with the KRB in the data base being loaded.

The name of the device DR reflects the name in the DCB that is being loaded. An empty slot in the table of DCB addresses in the resident data base will be made to point to this DCB.

The LOAD routines need to find the correct KRB in the data base being loaded. A global label of the form $cca (where cc is the controller name and a is the slot, or controller number) must define the start of the KRB(s) being loaded. Thus, the loadable data base for the example above must contain the labels $RHB and $RHC, which are the KRB names. The address(es) of the label(s) is loaded into the appropriate slot in the CTB table of KRB addresses.

In summary, then, the /CTB syntax on the LOAD command combined with the global label(s) allows the LOAD routines to link a driver data base being loaded with a currently resident driver data base. The KRB(s) being loaded are incorporated in the resident data base and the DCB being loaded is connected to the common interrupt table.

CHAPTER 6

DEBUGGING A USER-SUPPLIED DRIVER

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

1. Crash dump analysis support routine (CDA)

2. Executive debugging tool (XDT)

You need not select any of this software during system generation. If, however, you do require the facilities they offer, you can select them for incorporation in your system. The following sections describe the features and use of each debugging aid.

6.1 CRASH DUMP ANALYSIS SUPPORT ROUTINE

The crash dump analysis (CDA) support routine prints the following message on a notification device specified at system generation:

```
CRASH - CONT WITH SCRATCH MEDIA ON device mnemonic
```

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

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

6.2 THE EXECUTIVE DEBUGGING TOOL

An interactive debugging tool aids in debugging Executive modules, I/O drivers, and interrupt service routines. This debugging aid, called XDT, is a version of RSX-11 ODT. Including XDT in a system with Executive data space support does not reduce the size of pool space that the system can have. XDT occupies physical address space but does not take up any Executive virtual data address space. XDT also does not interfere with user-level RSX-11 ODT, which can be used with any number of tasks while you are debugging your driver with XDT.

You can include XDT in a system during the "Choosing Executive Options" section of system generation when the following question is asked:

```
Do You Want To Include XDT?  [Y/N D:N]
```

If you answer Y, XDT is linked into the Executive image when you build the Executive.

6.2.1 XDT Commands

XDT commands are generally compatible with RSX-11 ODT commands, which are described in the IAS/RSX-11 ODT Reference Manual. That manual, together with the discussion in Section 6.2 in this manual, can be used as a guide to XDT operation on RSX-11M-PLUS.

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) ODT command has been changed for XDT. The X command causes a jump to the crash dump routine.

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

XDT includes both Instruction space and Data space address referencing. The following commands control the current address referencing:

I	sets address references to Instruction space
D	sets address references to Data space

When XDT starts up, the default condition is that address references are to Data space.

6.2.2 XDT Start Up

When you bootstrap a system that includes XDT, the normal system startup transfers control to XDT, which identifies itself at the system console terminal with the following message:

```
XDT: <system name and version>
```

If no errors were encountered, the identification message is followed by the prompt:

```
XDT>
```

The following are the register conditions when XDT starts:

```
R0 = CSR address of the bootstrap device
R1 = LBN of the system image
R2 = LBN of the system image
R3 = physical unit number of the load device
R4 = ASCII name of the load device
R5 = total number of blocks read from the system image
```

XDT runs entirely at priority level 7.

You can set breakpoints at this time, and then give a G command, passing control to the Executive initialization module INITL. Whenever control reaches a breakpoint, a printout similar to that of RSX-11 ODT occurs.

If INITL encounters an error condition, it prints an error message preceded by a prefix telling whether the condition is a warning or fatal. If the condition is a warning, XDT has control. You can set breakpoints to establish control or type the P command to proceed. If the condition is fatal, the processor halts. You must correct the condition before you can rebootstrap your system.

6.2.3 XDT Restrictions

On some types of systems, the following restrictions exist on the use of XDT when it is first entered:

1. All systems

   Some data structures are not yet initialized. The secondary pool is not set up and the console terminal and the bootstrapped device are not on-line.

2. Systems with Kernel data space support

   Data space mapping is not yet set up. Certain Executive locations that the Task Builder could not resolve are not initialized. (The RH common interrupt table address ($RHTBX) does not contain the RH common interrupt routine address ($RHALT).)

To proceed when you encounter such restrictions on your system, at the initial XDT prompt you should first set a breakpoint near the end of the INITL module (after the routine that sets up the data structures). Then, after you proceed and XDT encounters the breakpoint near the end of INITL, use XDT to examine locations in the Executive and to set more Executive breakpoints.

On a multiprocessor system, you should be aware of the following conditions:

1. When you initially place a processor on-line, XDT does not prompt from that processor unless you have set the processor's bit in the $XDTPR word.

2. XDT does not handle multiprocessor-specific conditions. You cannot set processor-specific breakpoints nor can you easily examine other processors' low memory context.

3. Under certain circumstances (such as when Data space is not yet set up), setting a breakpoint in shared Instruction space may eventually cause a trap on a processor other than the one on which you set the breakpoint. Consequently, because the processor encountering the breakpoint does not have that breakpoint in its XDT tables, XDT generates a breakpoint error message (BE:) rather than a breakpoint message.

4. All processors are locked out of the Executive while XDT is being executed by one of the other processors.

6.2.4 XDT General Operation

A forced entry to XDT can be executed at any time from a privileged terminal by means of the MCR Breakpoint (BRK) command. Thus, if your system has no XDT restrictions as described in Section 6.2.3, the normal procedure is to type G when the system is bootstrapped without setting any breakpoints. When it becomes necessary to use XDT, the MCR Breakpoint command is executed, causing a forced breakpoint. XDT then prints on the console terminal:

```
BE:xxxxxx
```

This message is followed by the prompt:

```
XDT>
```

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

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

6.2.5 XDT and Debugging a User-Supplied Driver

Using XDT to debug a loadable 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.

To avoid this T-bit error, assemble the driver with an embedded BPT instruction, or use either the ZAP utility or the MCR OPEN function to set the breakpoint by replacing a word of code with the BPT instruction. You can use the OPEN command in the following way to access the driver:

```
>OPE nnnn/DRV=xx:
```

where nnnn matches the address in the driver map listing and xx is the device mnemonic. (Write down the instruction that you replace with the BPT instruction.)

When control reaches a breakpoint set in the driver, XDT prints:

```
BE:xxxxxx
XDT>
```

Recover as follows:

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.

NOTE

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

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 host hardware has faulted.

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

6.3.1 Immediate Servicing

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

1. If XDT is included, an unintended trap to XDT occurs.
2. The system displays text indicating a crash has occurred and halts.
3. The system halts but displays nothing.
4. The system is in an unintended loop.

The following discussions assume the existence of a system built with at least one debugging aid.

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** - The trap may or may not be intended (for example, a previously set breakpoint). If it is not intended, type the X command, causing XDT to exit to location 40(8), from which the CDA support routine will be invoked. If, however, you have some idea of the source of the problem (for example, a recent coding change), then you may use XDT to examine pertinent data structures and code.

6.3.1.2 **The System Reports a Crash** - If the text displayed on the console terminal consists of output from the CDA support routine, follow the procedure for obtaining and formatting a memory dump as outlined in the RSX-11M/M-PLUS Crash Dump Analyzer Reference Manual.

6.3.1.3 **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). The procedure depends on the particular PDP-11 processor. Consult the appropriate PDP-11 processor handbook for details.

After preserving the PS and PC, invoke your resident debugging aid: enter 40(8) in the switch register, press LOAD ADDR, and then press START. The contents of 40(8) cause the invocation of the CDA support routine.

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

1. Halt the processor.

2. Record the PC, the PS, and any pertinent device registers, as in Section 6.3.1.3.

You may then want to step through a number of instructions in an attempt to locate the loop. For this attempt to be meaningful you must first disable the system clock. Proceed as follows:

1. Examine the contents of word 777546 (if your system has a line-frequency clock) or word 772540 (if your system has a programmable clock).

2. Clear bit 6 in this word and redeposit the word.

NOTE

> Until you reenable the clock, some system operations do not work because they are waiting for time. You can type and the system echoes typed characters. You can input MCR commands.

After trying to locate the loop and reenabling the clock, transfer to location 40(8) as in Section 6.3.1.3.

6.3.2 Pertinent Fault Isolation Data

Before you attempt to locate the fault, you should dump system common (SYSCM). SYSCM contains a number of critical pointers and listheads. CDA always dumps the SYSCM area. In addition, you should dump the dynamic storage region (system pool and, if it exists, the ICB pool) and the device tables. The device tables are in the module SYSTB.

At this point, you have the following data:

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

These data represent a minimal requirement for effectively tracing the fault.

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)

The task header and its associated pointers are described below.

The location of the task header and the contents of its associated pointers vary according to whether the task has an external header. A task with an external header has its header attached in a physically contiguous and numerically lower location in memory. A task with a nonexternal header has its header located in Executive pool space. Therefore, a header in Executive pool is a pool-resident header, and a header adjacent to the task is a non-pool-resident header.

Figure 6-1 shows the interaction of header pointers for both pool-resident and non-pool-resident headers. For a pool-resident task header, $HEADR, $SAHPT, and $SAVSP all point to the first word of the task header. This word also 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 RSX-11M/M-PLUS Task Builder Manual.

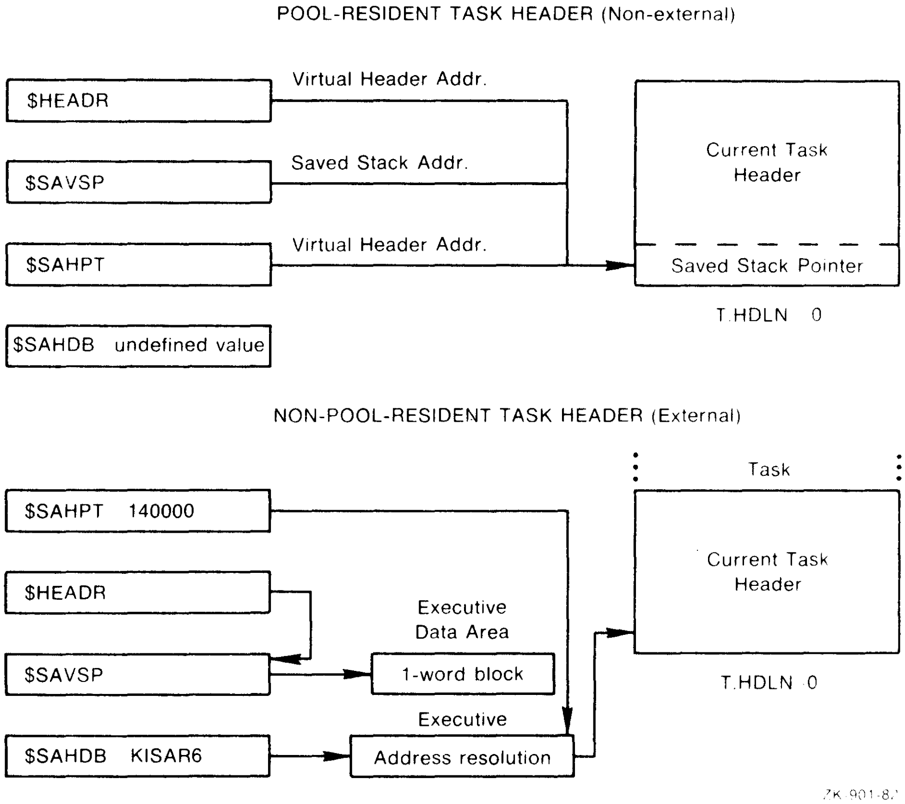


Figure 6-1: Interaction of Task Header Pointers

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).

The four pointers associated with the header are:

- $HEADR
- $SAVSP
- $SAHPT
- $SAHDB

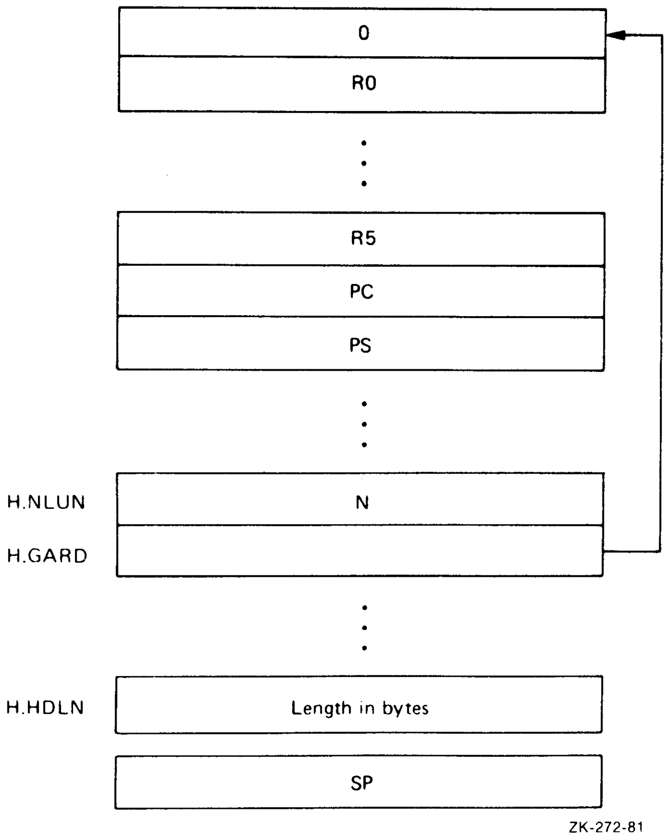


Figure 6-2: Task Header

The pointers associated with a pool-resident 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 - Contains an unknown value.

The pointers associated with a non-pool-resident header are described next:

$HEADR - Points to the pointer for the saved stack pointer, $SAVSP.

$SAVSP - Points to a 4-word block in the Executive data area.

$SAHPT - Contains the octal value of 140000 that is to be used with $SAHDB to resolve the address of the task's header. $SAHPT always contains 140000 in this case.

$SAHDB - Contains the value in KISAR6, which is a 32-word block-offset to be used with the value in $SAHPT to resolve the address of the task's header.

6.4.1 Tracing Faults Using the Executive Stack and Register Dump

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

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

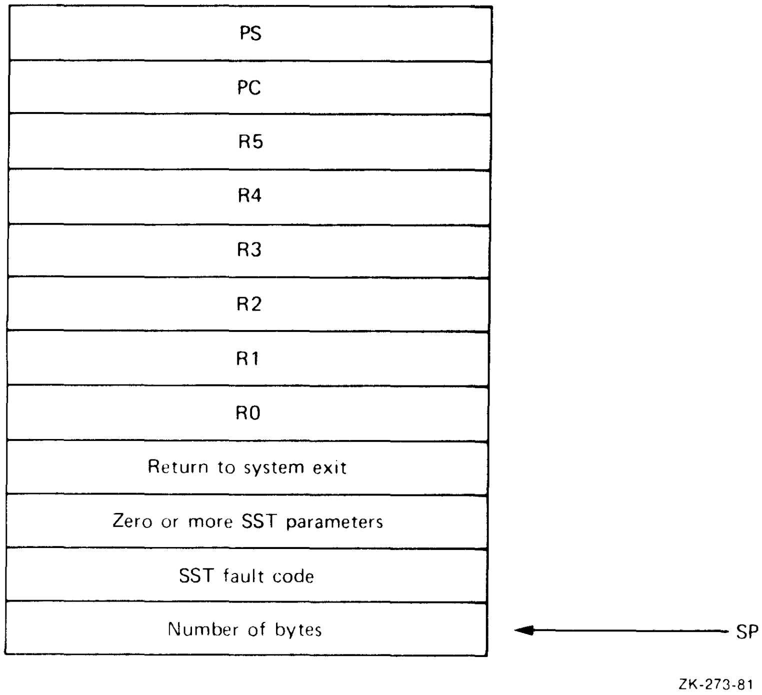


Figure 6-3: Stack Structure: Internal SST Fault

The fault codes are:

```
0        ;ODD ADDRESS AND TRAPS TO 4
2        ;MEMORY PROTECT VIOLATION
4        ;BREAK POINT OR TRACE TRAP
6        ;IOT INSTRUCTION
10       ;ILLEGAL OR RESERVED INSTRUCTION
12       ;NON RSX EMT INSTRUCTION
14       ;TRAP INSTRUCTION
16       ;11/40 FLOATING POINT EXCEPTION
20       ;SST ABORT-BAD STACK
22       ;AST ABORT-BAD STACK
24       ;ABORT VIA DIRECTIVE
```

```
26        ;TASK LOAD READ FAILURE
30        ;TASK CHECKPOINT READ FAILURE
32        ;TASK EXIT WITH OUTSTANDING I/O
34        ;TASK MEMORY PARITY ERROR
```

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

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

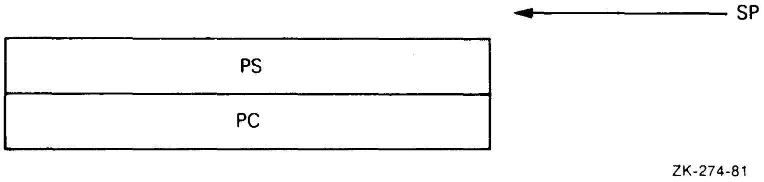


Figure 6-4: Stack Structure: Abnormal SST Fault

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.

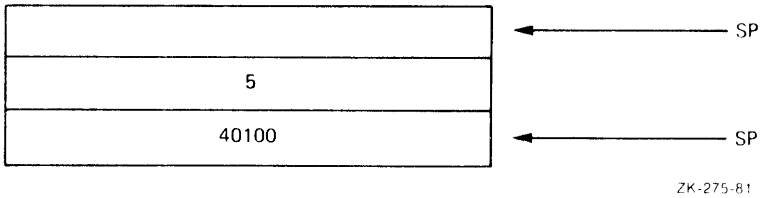


Figure 6-5: Stack Structure: Data Items on Stack

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.

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

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. If XDT has been included in the system, the deallocation routine checks for bad deallocation and crashes the system if it occurs.

6.5 REBUILDING AND REINCORPORATING A LOADABLE DRIVER

After correcting and assembling the driver source and updating the Executive object library, simply unload the old version, using the MCR command UNLOAD, task build the new one, and load it using the LOAD command. The commands for the assembler, Librarian, and Task Builder are shown in Section 5.2.

Once loaded, the data base is not removed by the UNLOAD command. If the data base is in error and cannot be patched, correct its source, reassemble it, update the Executive object library RSX11M.OLB 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 THE ADDRESS DOUBLE WORD

RSX-11M-PLUS 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 RSX-11M-PLUS 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 virtual address. The result becomes the first word of the address doubleword. It represents the nth 32-word block in a memory viewed as a collection of 32-word blocks. Note that at the time the address doubleword is computed, the user'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.

7.3 DRIVERS FOR NPR DEVICES USING EXTENDED MEMORY

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

Non-Extended memory NPR devices on the PDP-11 processor must perform I/O transfers by using UNIBUS Mapping Registers (UMRs) as described in the <u>PDP-11 Processor Handbook</u>. One UMR is required for each 4K words involved in the transfer -- as specified by the contents of U.CNT in the UCB. When multiple UMRs are required for a transfer, they must be contiguous.

A driver can be assigned UMRs through any one of three procedures:

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

NOTE

> In large systems, use of the procedures above to hold UMRs for longer periods than necessary can result in the blocking of other drivers and a reduction in system throughput.

7.3.1 Calling $STMAP and $MPUBM or $STMP1 and $MPUB1

To obtain UMRs through use of the $STMAP and $MPUBM or the $STMP1 and $MPUB1 routines, a driver must:

- Have available six words for a mapping register assignment block in the 22-bit working storage area of the device's controller request block (KRB). The end of this area is accessed by adding the contents of K.OFF to the address at K.CSR. If the driver uses $STMP1 and $MPUB1, it must also have available a 10-word block

- Call the routine $STMAP or $STMP1 (Set Up UNIBUS Mapping Address) after getting the I/O packet

- Call the routine $MPUBM or $MPUB1 (Map UNIBUS to Memory) before initiating a transfer

These requirements are detailed in the following three subsections. Note that these routines are only required when the driver is performing a data transfer.

7.3.1.1 **Allocating a Mapping Register Assignment Block** - The controller request block (KRB) of an NPR device requires a 6-word mapping register assignment block located in the 22-bit working storage area. It does not have to be initialized. Any initial contents are simply overwritten.

The following example shows the allocation of a mapping register assignment block.

```
    .BLKW   M.LGTH  ;UMR WORK AREA
```

If the driver does not support parallel NPR operations requiring UMR mapping, it calls $STMAP and $MPUBM. If the driver supports parallel NPR operations requiring UMR mapping, it must call $STMP1 and $MPUB1. In the latter situation, the six additional words in the 22-bit working storage area are not used but must still be present. In addition, the driver data base must provide a 10-word mapping register assignment block for each data transfer to be mapped as specified in the description of $STMP1 later in Section 7.4.31.

7.3.1.2 **Calling $STMAP or $STMP1** - In the coding at the initiator entry point, after the call to $GTPKT, the NPR-device driver must call the routine $STMAP or $STMP1. These routines dynamically allocate required UMRs. If UMRs are not available immediately, the driver is blocked. Such blocking, if it occurs, is completely transparent to the driver. The driver resumes processing at fork level when the UMRs have been allocated. The register returns are absolutely identical whether or not blocking has occurred.

$STMAP or $STMP1 stores into U.BUF and U.BUF+2 (in the UCB) a UNIBUS address that causes the appropriate UMR to be selected for mapping the transfer. The call to $STMAP or $STMP1 must be conditionalized on M$$EXT.

7.3.1.3 **Calling $MPUBM or $MPUB1** - Before executing the transfer, the driver must call $MPUBM or $MPUB1. These routines get the buffer's

22-bit physical address and load the UNIBUS mapping registers so that transfers are mapped directly to the task's space. The call to $MPUBM or $MPUB1 must be conditionalized on M$$EXT.

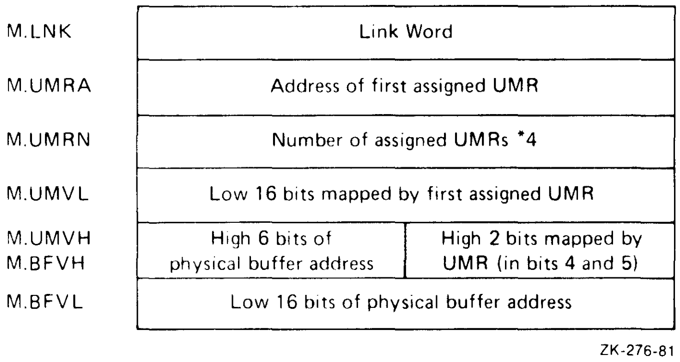
If the driver calls $STMAP and $MPUBM, the UMRs allocated to it are deallocated during the call to $IODON or $IOALT. If the driver calls $STMP1 and $MPUB1, it must call $DEUMR to deallocate any allocated UMRs before calling $IODON or $IOALT.

7.3.2 Calling $ASUMR and $DEUMR

Use of the procedure described in Section 7.3.3 assures that UMRs are always allocated. However, a driver may not require UMRs to be allocated all of the time, and yet require UMRs for periods of time longer than the normal time-frame between $GTPKT and $IODON (or $IOALT). In such cases, there is a third procedure for allocating UMRs.

Through use of the Executive routines $ASUMR and $DEUMR, a driver can dynamically allocate, retain over a desired time-frame, and deallocate UMRs. Refer to Section 7.4 for descriptions of the $ASUMR and $DEUMR routines.

Similar to the $STMAP/$MPUBM procedure, the use of $ASUMR and $DEUMR also requires the allocation of a 6-word mapping register assignment block. In this instance, however, the block must not be located in the 22-bit working storage area. $IODON or $IOALT, when called, will attempt to deallocate the UMRs of a block found in the 22-bit working storage area. To avoid this deallocation, the mapping register assignment block could be dynamically allocated from the pool. Figure 7-1 details the format of the 6-word block.



M.LNK	Link Word	
M.UMRA	Address of first assigned UMR	
M.UMRN	Number of assigned UMRs *4	
M.UMVL	Low 16 bits mapped by first assigned UMR	
M.UMVH M.BFVH	High 6 bits of physical buffer address	High 2 bits mapped by UMR (in bits 4 and 5)
M.BFVL	Low 16 bits of physical buffer address	

Figure 7-1: Mapping Register Assignment Block

7.3.3 Statically Allocating UMRs During System Generation

UMRs can be statically assigned during system generation. The system generation procedure defines the symbol N$$UMR equal to a fixed number of UMRs, multiplied by 4, that are statically assigned to the system. Before assembling the Executive, the user can cause the static allocation of an additional number of UMRs by editing the Executive assembly prefix file RSXMC.MAC. The value of the symbol N$$UMR can then be increased to represent the additional number of desired UMRs multiplied by 4.

The Executive assembly prefix file RSXMC.MAC further defines the following three symbols, which describe the first UMR statically allocated during system generation:

U$$MRN The I/O page address of the first UMR register available for allocation to the user

U$$MLO The low-order 16 bits of the 18-bit UNIBUS address mapped by this UMR

U$$MHI The high-order 2 bits of the 18-bit UNIBUS address; these 2 bits are in bit positions 4 and 5

These three symbols are not used by the system itself. They are available for the user's information.

7.4 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 7-1 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 7-1
Summary of Executive Service Calls for Drivers

Routine Name	Location in UFD [11,10]	Function
$ACHKB	EXSUB	Adress check for byte-aligned buffers
$ACHCK	EXSUB	Address check for word-aligned buffers
$ALOCB	CORAL	Alocate core buffer
$ASUMR	MEMAP	Assign UNIBUS mapping registers
$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
$DEUMR	MEMAP	Deassign UNIBUS mapping registers
$DVMSG	IOSUB	Device message output
$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
$INTSV	SYSXT	Interrupt save and restore
$INTXT	SYSXT	Interrupt exit

(continued on next page)

Table 7-1 (Cont.)
Summary of Executive Service Calls for Drivers

Routine Name	Location in UFD [11,10]	Function
$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
$MPUBM	MEMAP	Map UNIBUS memory
$MPUB1	MEMAP	Alternate $MPUBM entry for parallel operations
$PTBYT	BFCTL	Put byte
$PTWRD	BFCTL	Put word
$QINSP	QUEUE	Queue insertion by priority
$RELOC	MEMAP	Relocate address
$RELOP	MEMAP	Relocate UNIBUS physical address
$REQUE	IOSUB	Queue kernel AST to task
$REQU1	IOSUB	Queue kernel AST to task
$STMAP	MEMAP	Set up UNIBUS mapping address
$STMP1	MEMAP	Alternate $STMAP entry for parallel operations
$TSPAR	REQSB	Test if partition memory resident for kernel AST
$TSTBF	IOSUB	Test for I/O buffering

$ACHKB
$ACHCK

7.4.1 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 8.3 for an example.

Calling Sequences:

```
    CALL       $ACHKB
```

or

```
    CALL       $ACHCK
```

Description:

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

Notes:

In RSX-11M-PLUS Version 2.0, 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 7.4.6 for a description of $CKBFB/$CKBFW and Section 8.3 for an example.

$ALOCB

7.4.2 Allocate Core Buffer

This routine is in the file CORAL.

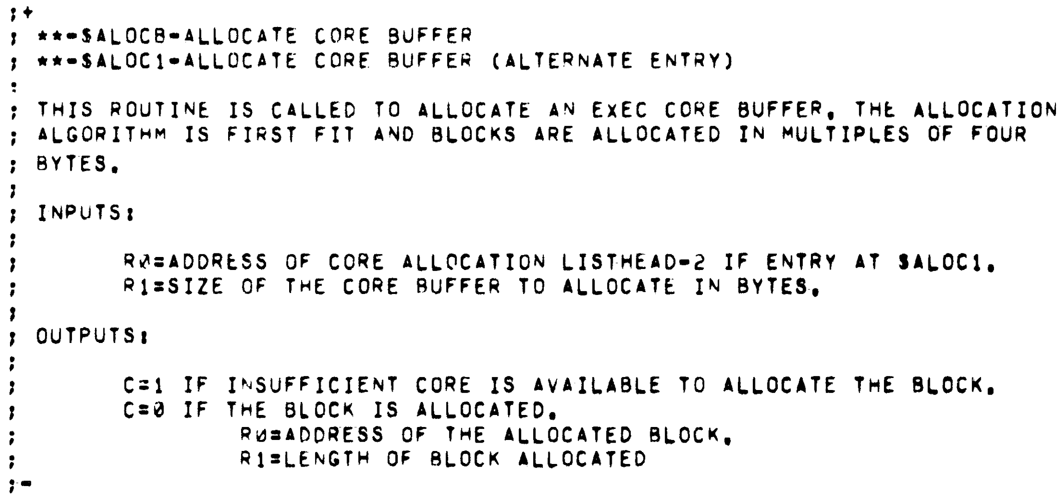
Calling Sequences:

    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
;-
```

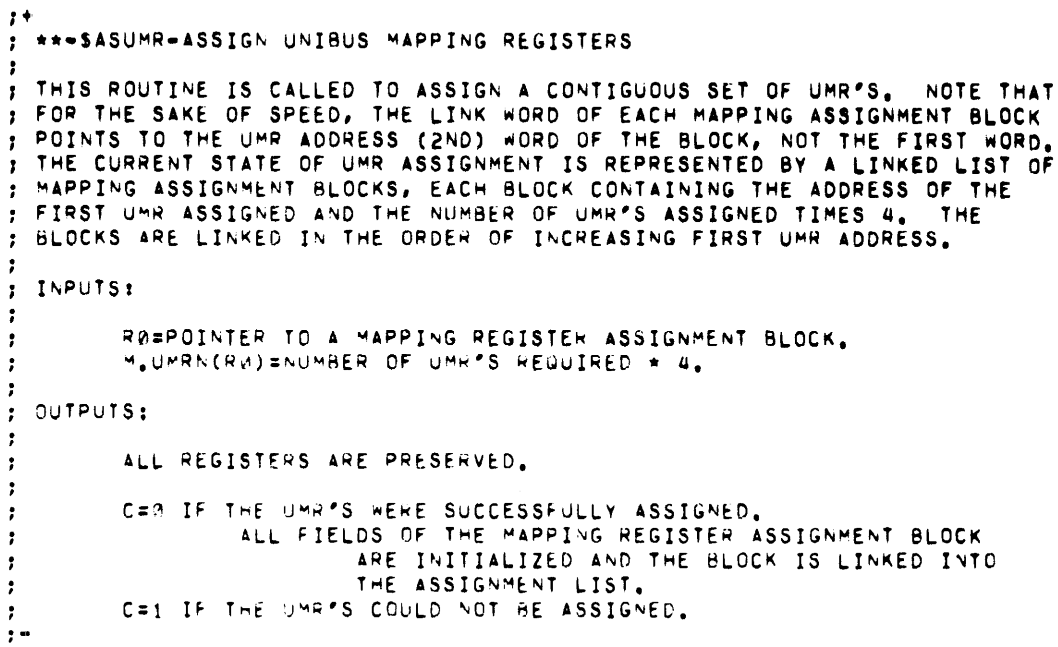
7.4.3 Assign UNIBUS Mapping Registers

This routine is in the file MEMAP. It is used only for PDP-11/70 NPR devices requiring UNIBUS Mapping Registers when 22-bit memory addressing is enabled. Normally, it is not called directly by an I/O driver. Rather, it is called from within the $STMAP routine. Refer to Section 7.3 for a discussion.

Calling Sequence:

    CALL $ASUMR

Description:



```
;+
; **-$ASUMR-ASSIGN UNIBUS MAPPING REGISTERS
;
; THIS ROUTINE IS CALLED TO ASSIGN A CONTIGUOUS SET OF UMR'S.  NOTE THAT
; FOR THE SAKE OF SPEED, THE LINK WORD OF EACH MAPPING ASSIGNMENT BLOCK
; POINTS TO THE UMR ADDRESS (2ND) WORD OF THE BLOCK, NOT THE FIRST WORD.
; THE CURRENT STATE OF UMR ASSIGNMENT IS REPRESENTED BY A LINKED LIST OF
; MAPPING ASSIGNMENT BLOCKS, EACH BLOCK CONTAINING THE ADDRESS OF THE
; FIRST UMR ASSIGNED AND THE NUMBER OF UMR'S ASSIGNED TIMES 4.  THE
; BLOCKS ARE LINKED IN THE ORDER OF INCREASING FIRST UMR ADDRESS.
;
; INPUTS:
;
;       R0=POINTER TO A MAPPING REGISTER ASSIGNMENT BLOCK.
;       M.UMRN(R0)=NUMBER OF UMR'S REQUIRED * 4.
;
; OUTPUTS:
;
;       ALL REGISTERS ARE PRESERVED.
;
;       C=0 IF THE UMR'S WERE SUCCESSFULLY ASSIGNED.
;               ALL FIELDS OF THE MAPPING REGISTER ASSIGNMENT BLOCK
;                       ARE INITIALIZED AND THE BLOCK IS LINKED INTO
;                       THE ASSIGNMENT LIST.
;       C=1 IF THE UMR'S COULD NOT BE ASSIGNED.
;-
```

$BLKCK
$BLKC1
$BLKC2

7.4.4 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.
;-
```

7.4.5 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 4K-63).
;-
```

$CKBFI
$CKBFR
$CKBFW
$CKBFB

7.4.6 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
;-
```

7.4.7 Clock Queue Insertion

This routine is in the file QUEUE.

Calling Sequence:

```
     CALL       $CLINS
```

Description:

```
;+
; **-$CLINS-CLOCK QUEUE INSERTION
;
; THIS ROUTINE IS CALLED TO MAKE AN ENTRY IN THE CLOCK QUEUE. THE ENTRY
; IS INSERTED SUCH THAT THE CLOCK QUEUE IS ORDERED IN ASCENDING TIME.
; THUS THE FRONT ENTRIES ARE MOST IMMINENT AND THE BACK LEAST.
;
; INPUTS:
;
;       R0=ADDRESS OF THE CLOCK QUEUE ENTRY CORE BLOCK.
;       R1=HIGH ORDER HALF OF DELTA TIME.
;       R2=LOW ORDER HALF OF DELTA TIME.
;       R4=REQUEST TYPE.
;       R5=ADDRESS OF REQUESTING TCB OR REQUEST IDENTIFIER.
;
; OUTPUTS:
;
;       THE CLOCK QUEUE ENTRY IS INSERTED IN THE CLOCK QUEUE ACCORDING
;       TO THE TIME THAT IT WILL COME DUE.
;
; NOTE:
;       ON MULTIPROCESSOR SYSTEMS, A REQUEST WITH TYPE C.SYST!100000
;       WILL BE EXECUTED ON A PRATICULAR UNIBUS RUN, WITH URM
;       SPECIFIED IN C.URM. TYPE C.CYST REQUESTS ON MP SYSTEMS ARE
;       DEFAULTED TO RUN ON ANY UNIBUS RUN, WHICH IN PRACTICE WILL
;       RESULT IN THE REQUEST EXECUTING ON THE CPU WHICH OWNS THE
;       CLOCK. ($CKURM)
;-
```

$CVLBN

7.4.8 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
;-
```

7.4.9 Deallocate Core Buffer

This routine is in the file CORAL.

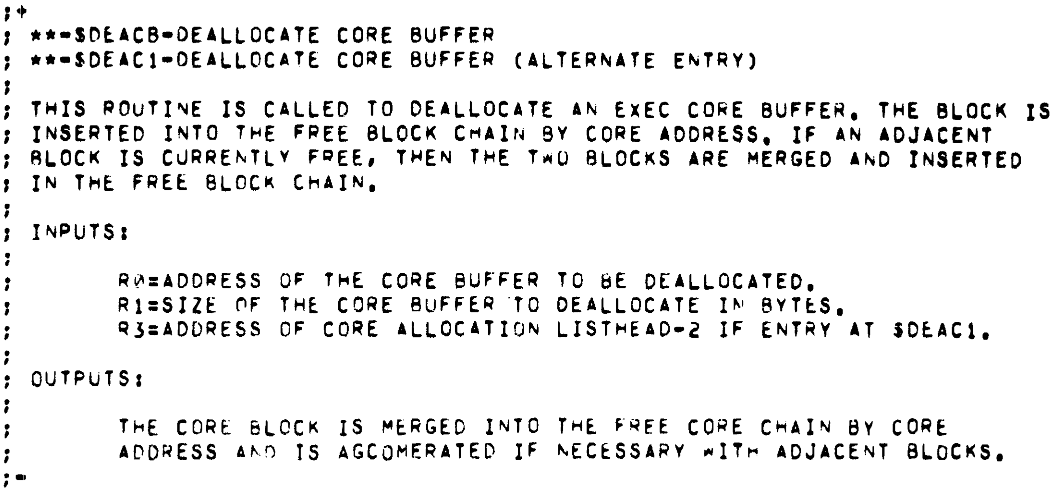
Calling sequences:

```
     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 AGCOMERATED IF NECESSARY WITH ADJACENT BLOCKS.
;-
```

$DEUMR

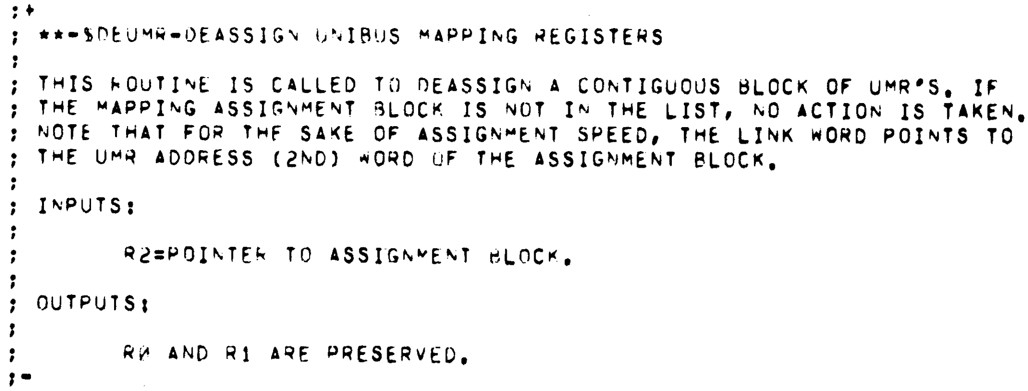
7.4.10 Deassign UNIBUS Mapping Registers

This routine is in the file MEMAP. It is used only for NPR devices requiring UNIBUS Mapping Registers when 22-bit addressing is enabled. Normally, it is not called directly by an I/O driver. Rather, it is called from within the $IODON routine. Refer to Section 7.3 for a discussion.

Calling Sequence:

```
     CALL        $DEUMR
```

Description:



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

7.4.11 Device Message Output

Device Message Output is in file IOSUB.

Calling Sequence:

```
     CALL       $DVMSG
```

Description:

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

Note:

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

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

or

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

$FORK

7.4.12 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:

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

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

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

7.4.13 Forkl

Forkl is in the file SYSXT. A driver calls $FORKl 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        $FORKl
```

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 PRESERVED FOR CALLERS CALLER.
;-
```

Notes:

1. A 5-word fork block is required for calls to $FORKl.

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 $FORKl is called.

$GTBYT

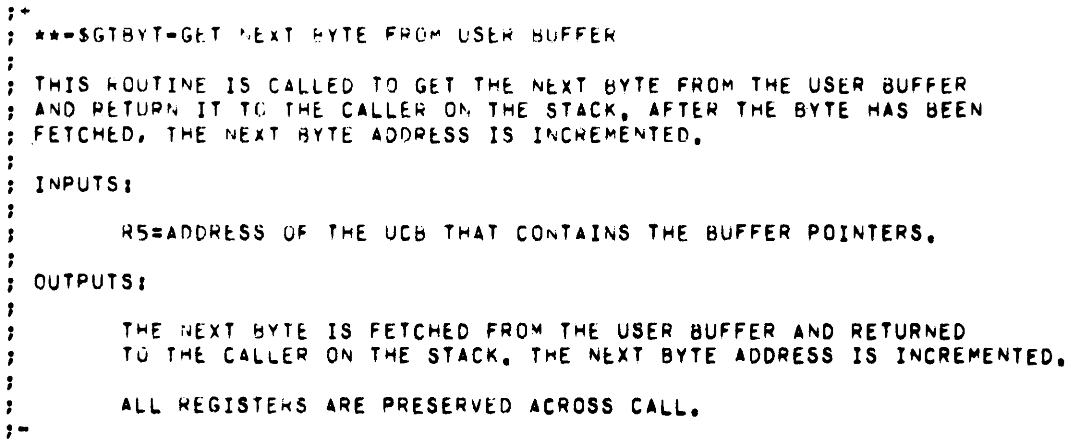
7.4.14 Get Byte

Get Byte is in the file BFCTL. Get Byte manipulates words U.BUF and U.BUF+2 in the UCB.

Calling sequence:

```
     CALL        $GTBYT
```

Description:



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

7.4.15 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 4.3. Usage of $GSPKT is described briefly in Section 1.4.7.

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.
```

$GTPKT
$GSPKT (Cont.)

```
;
;               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.
;-
```

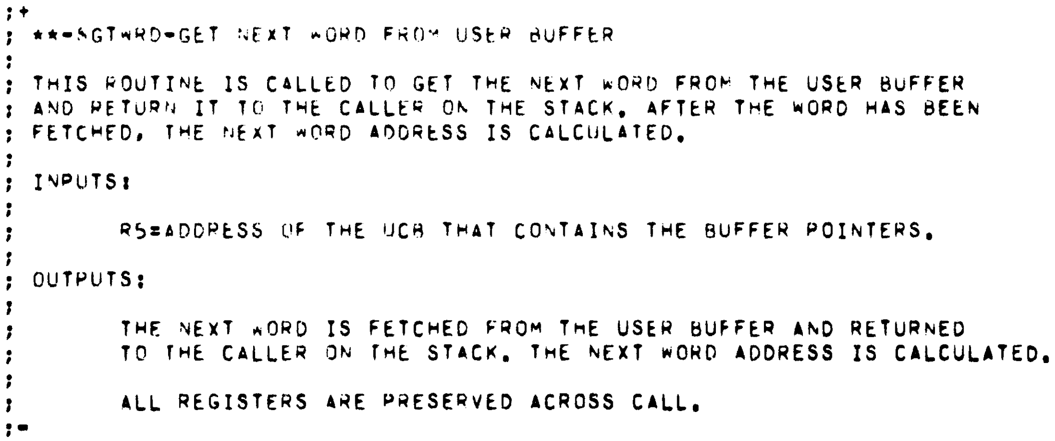
7.4.16 Get Word

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.
;-
```

$INIBF

7.4.17 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.
     ;-
```

7.4.18 Interrupt Save

Interrupt Save is in the file SYSXT. The recommended way to use $INTSV is to use the INTSV$ macro call described in Section 4.3.

Calling Sequence:

```
CALL       $INTSV,PRn
```

n has a range of 0-7.

Description:

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

Note:

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

$INTXT

7.4.19 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:
;
;       @(SP)=INTERRUPT SAVE RETURN ADDRESS.
;
; OUTPUTS:
;
;       A RETURN TO INTERRUPT SAVE IS EXECUTED.
;-
```

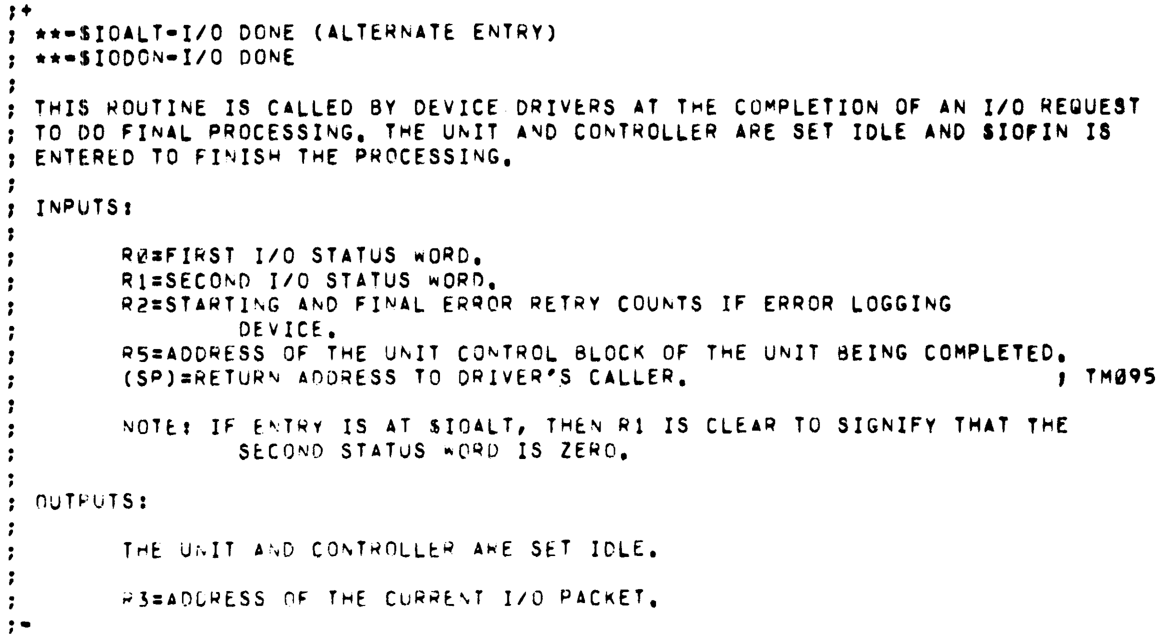
7.4.20 I/O Done Alternate Entry and I/O Done

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.                       ; TM095
;
;       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.
;
;       R3=ADDRESS OF THE CURRENT I/O PACKET.
;-
```

Note:

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

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

$IOFIN

7.4.21 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 8.3 for an example) calls I/O Finish if the driver finds an error while preprocessing the I/O packet.

Calling Sequence:

```
     CALL        $IOFIN
```

Description:

```
;+
; **-$IOFIN-I/O FINISH
;
; THIS ROUTINE IS CALLED TO FINISH I/O PROCESSING IN CASES WHERE THE UNIT AND
; CONTROLLER ARE NOT TO BE DECLARED IDLE. 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.
;-
```

$MPUBM

7.4.22 Map UNIBUS to Memory

This routine is in the file MEMAP. It is used only for NPR devices requiring UNIBUS Mapping Registers when 22-bit memory addressing is enabled. See Section 7.3 for a discussion.

Calling Sequence:

```
    CALL        $MPUBM
```

Description:

```
;+
; **-$MPUBM-MAP UNIBUS TO MEMORY
;
; THIS ROUTINE IS CALLED BY UNIBUS NPR DEVICE DRIVERS TO LOAD THE
; NECESSARY UNIBUS MAP REGISTERS TO EFFECT A TRANSFER TO MAIN MEM-
; ORY ON AN 11/70 PROCESSOR WITH EXTENDED MEMORY.
;
; INPUTS:
;
;       R4=ADDRESS OF DEVICE SCB.
;       R5=ADDRESS OF DEVICE UCB.
;
; OUTPUTS:
;
;       THE UNIBUS MAP REGISTERS NECESSARY TO EFFECT THE  TRANSFER
;       ARE LOADED.
;
; NOTE: REGISTER R3 IS PRESERVED ACROSS CALL.
;-
```

$MPUB1

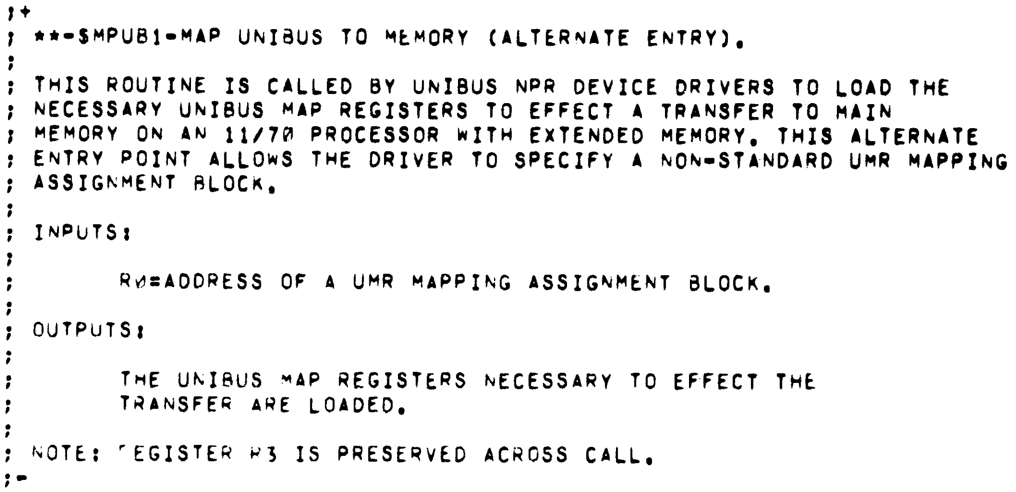
7.4.23 Map UNIBUS to Memory (Alternate Entry)

This routine is in file MEMAP. It is used only for NPR devices that require UNIBUS Mapping Registers when 22-bit memory addressing is enabled and for support parallel operations.

Calling Sequences:

```
CALL       $MPUB1
```

Description:



```
;+
; **-$MPUB1-MAP UNIBUS TO MEMORY (ALTERNATE ENTRY).
;
; THIS ROUTINE IS CALLED BY UNIBUS NPR DEVICE DRIVERS TO LOAD THE
; NECESSARY UNIBUS MAP REGISTERS TO EFFECT A TRANSFER TO MAIN
; MEMORY ON AN 11/70 PROCESSOR WITH EXTENDED MEMORY. THIS ALTERNATE
; ENTRY POINT ALLOWS THE DRIVER TO SPECIFY A NON-STANDARD UMR MAPPING
; ASSIGNMENT BLOCK.
;
; INPUTS:
;
;       R0=ADDRESS OF A UMR MAPPING ASSIGNMENT BLOCK.
;
; OUTPUTS:
;
;       THE UNIBUS MAP REGISTERS NECESSARY TO EFFECT THE
;       TRANSFER ARE LOADED.
;
; NOTE: REGISTER R3 IS PRESERVED ACROSS CALL.
;-
```

7.4.24 Put Byte

Put Byte is in the file BFCTL. Put Byte manipulates words U.BUF and U.BUF+2 in the UCB.

Calling Sequence:

```
     CALL       $PTBYT
```

Description:

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

$PTWRD

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

$QINSP

7.4.26 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 8.3 for an example.

Calling Sequence:

```
     CALL       $QINSP
```

Description:

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

$RELOC

7.4.27 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 8.3 for an example.

Calling Sequence:

```
     CALL        $RELOC
```

Description:

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

7.4.28 Relocate UNIBUS Physical Address

This routine is in the file MEMAP.

Calling Sequence:

```
     CALL        $RELOP
```

Description:

```
;+
; **-$RELOP-RELOCATE UNIBUS PHYSICAL ADDRESS
;
; THIS ROUTINE RELOCATES A UNIBUS PHYSICAL ADDRESS TO A KISAR6
; BIAS AND DISPLACEMENT.
;
; INPUTS:
;
;       RØ=BYTE OFFSET FROM ADDRESS IN U.BUF+1 AND U.BUF+2
;       R4=SCB ADDRESS
;       R5=UCB ADDRESS
;               U.BUF+1(R5)=HIGH ORDER BITS OF PHYSICAL ADDRESS
;               U.BUF+2(R5)=LOW ORDER BITS OF PHYSICAL ADDRESS
;
; OUTPUTS:
;
;       KISAR6=CALCULATED BIAS (MAPPED SYSTEM)
;       R1=REAL ADDRESS OR DISPLACEMENT
;-
```

$REQUE
$REQU1

7.4.29 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.
;--
```

$STMAP

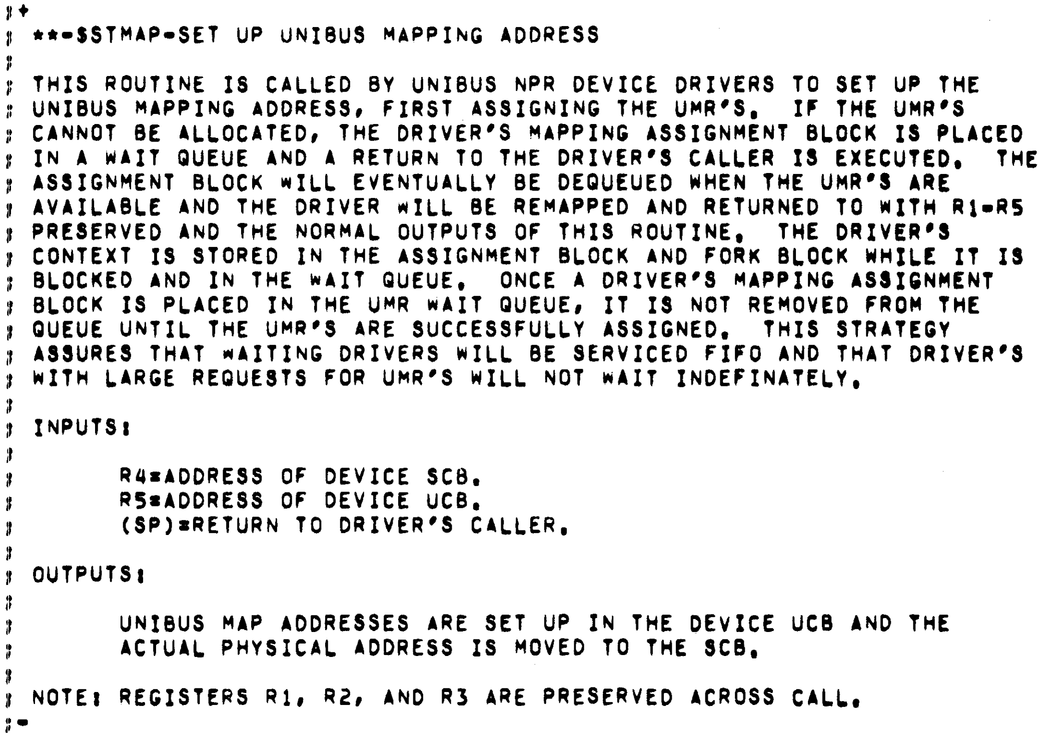
7.4.30 Set Up UNIBUS Mapping Address

This routine is in the file MEMAP. It is used only for NPR devices requiring UNIBUS Mapping Registers when 22-bit memory addressing is enabled. See Section 7.3 for a discussion.

Calling Sequence:

```
    CALL        $STMAP
```

Description:



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

Note:

This routine pushes the address of routine $DQUMR+2 onto the stack before returning to the caller. This precludes the use of the stack for temporary data storage by drivers when calling this routine.

$STMP1

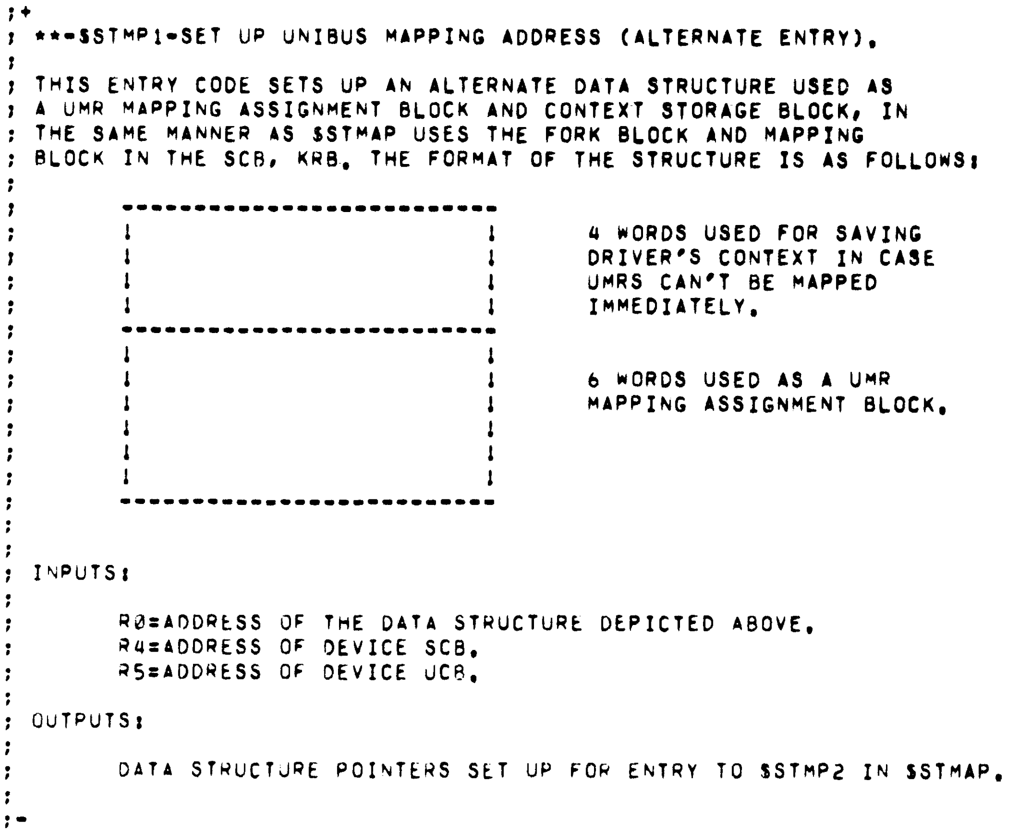
7.4.31 Set Up UNIBUS Mapping Address (Alternate Entry)

This routine is in file MEMAP. It is used only for NPR devices that require UNIBUS Mapping Registers when 22-bit memory addressing is enabled and for support parallel operations.

Calling Sequence:

```
     CALL        $STMP1
```

Description:



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

Note:

This routine pushes the address of routine $DQUMR+2 onto the stack before returning to the caller. This precludes the use of the stack for temporary data storage by drivers when calling this routine.

7.4.32 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

7.4.33 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.
     ;-
```

CHAPTER 8

SAMPLE DRIVER CODE

This chapter presents three sections of code. Sections 8.1 and 8.2 show the driver data base and driver code for a conventional driver. Section 8.3 gives a coding example from a driver that inhibits the automatic packet queuing in QIO processing so that it might address-check and relocate a special user buffer. Both of the sample drivers are in UFD [200,1] on the distribution kit.

In addition to the examples shown in this chapter, you should review the source code for one or more standard DIGITAL-supplied drivers. You should also examine the files SYSTB.MAC and XXTAB.MAC, which contain data structures created at system generation.

8.1 SAMPLE DRIVER DATA BASE

The following example shows the source code to create the data base for the driver that supports the DL device. The data base allows for one controller and one unit.

```
        .TITLE  DLTAB
        .IDENT  /09.0/
;
; SYSTEM TABLES
;
; 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
;
;
;
$DLDAT::
;
;                                               DL CTB
;
        .WORD   0               ; L.ICB
$CTB0:
        .WORD   $CTB1           ; L.LNK
        .ASCII  /DL/            ; L.NAM
        .WORD   .DC0            ; L.DCB
        .BYTE   1               ; L.NUM
        .BYTE   0               ; L.STS
$DLCTB::                        ; L.KRB
        .WORD   $DLA
;
;                                               DL DCB
;
$DLTBL=0                        ;LOADABLE DLDRV
$DLDCB::
.DC0:
        .WORD   .DC1            ; D.LNK
        .WORD   .DL0            ; D.UCB
        .ASCII  /DL/            ; D.NAM
        .BYTE   0,0             ; D.UNIT
        .WORD   DLND-DLST       ; D.UCBL
        .WORD   $DLTBL          ; D.DSP
        .WORD   177477,70,0,177200,377,0,0,377 ; D.MSK
        .WORD   0               ; D.PCB
;
;                                               DL UCB'S
;
DLST=.
        .WORD   0
.DL0::
        .WORD   .DC0
        .WORD   .-2
        .BYTE   UC.ALG!UC.NPR!UC.PWF!1,US.MNT
        .BYTE   0,US.OFL
        .WORD   DV.DIR!DV.MSD!DV.UMD!DV.F11!DV.MNT
        .WORD   0
        .WORD   50000
        .WORD   512.
        .WORD   $DL0
        .WORD   0,0,0,0,0,0,0,0
        .BYTE   40.,2.
        .WORD   512.
DLND=.
;
;                                               DLA  KRB
;
        .BYTE   PR5             ; K.PRI
        .BYTE   160/4           ; K.VCT
        .BYTE   0*2,0           ; K.CON, K.IOC
        .WORD   0!KS.OFL        ; K.STS
$DLA::  .WORD   174400          ; K.CSR
        .WORD   DLA-$DLA        ; K.OFF
        .BYTE   0,0             ; K.HPU
        .WORD   0               ; K.OWN
;
;       CONTIGUOUS   S C B   HERE FOR   DL
;
$DL0::  .WORD   0,.-2           ; S.LHD AND K.CRQ
        .WORD   0,0,0,0         ; S.FRK
        .WORD   0               ; S.KS5
        .WORD   0               ; S.PKT
        .BYTE   0               ; S.CTM
        .BYTE   4               ; S.ITM
        .BYTE   0               ; S.STS
        .BYTE   0               ; S.ST3
        .WORD   S2.LOG!S2.CON   ; S.ST2
        .WORD   $DLA            ; S.KRB
        .BYTE   7.              ; S.RCNT
        .BYTE   0               ; S.ROFF
        .WORD   0               ; S.EMB
        .BLKW   6               ; MAPPING ASSIGNMENT BLOCK
        .WORD   0               ; KE.RHB
DLA:
```

```
;
;
$DLEND::

.DC1 = 0                          ; END OF DCB LIST FOR DL:

$CTB1 = 0                         ; END OF CTB LIST FOR DL:

        .END
```

8.2 SAMPLE DRIVER CODE

The following example shows the source code for the DL driver. Comments beginning with ';;;' indicate that the instruction is being executed at a priority level greater than or equal to 5.

```
        .TITLE  DLDRV
        .IDENT  /01/

;
; RL11-RL01/02 DISK DRIVER
;

.MCALL  HWDDF$,PKTDF$
        HWDDF$                    ;DEFINE HARDWARE REGISTERS
        PKTDF$                    ;DEFINE I/O PACKET OFFSETS

;
; EQUATED SYMBOLS
;

RETRY=  8.                        ;CONTROLLER ERROR RETRY COUNT
RLBPT=  512.*20.                  ;BYTES PER SURFACE
RLSPU=  15.                       ;TIME TO SPIN UP

;
; RL11 DEVICE REGISTER OFFSETS
;

RLCS=   0                         ;CONTROL STATUS REGISTER
RLBA=   2                         ;BUS ADDRESS REGISTER
RLDA=   4                         ;DISK ADDRESS REGISTER
RLMP=   6                         ;MULTIPURPOSE REGISTER

;
; RLCS BIT ASSIGNMENTS
;

ERR=    100000                    ;COMPOSITE ERROR
DE=     040000                    ;DRIVE ERROR
NXM=    020000                    ;NONEXISTENT MEMORY
DLT=    010000                    ;DATA LATE
HNF=    010000                    ;HEADER NOT FOUND
DCK=    004000                    ;DATA CHECK ERROR
HCRC=   004000                    ;HEADER CRC ERROR
OPI=    002000                    ;OPERATION INCOMPLETE
DRDY=   1                         ;DRIVE READY
WCHK=   2                         ;WRITE CHECK FUNCTION
WRITE=  2                         ;WRITE OFFSET
GSTS=   4                         ;GET DRIVE STATUS FUNCTION
SEEK=   6                         ;SEEK FUNCTION
RDH=    10                        ;READ HEADERS FUNCTION
READ=   14                        ;READ DATA FUNCTION
```

```
IE=     100                         ;INTERRUPT ENABLE
CRDY=   200                         ;CONTROLLER READY

;
; RLDA STATUS CODES
;

MRK=    1                           ;MARKER BIT
STS=    2                           ;GET STATUS BIT
SN=     4                           ;SIGN BIT FOR SEEK
RST=    10                          ;DRIVE RESET BIT
HS=     20                          ;HEAD SELECT BIT FOR DIFFERENCE
REV=    200!MRK                     ;REVERSE SEEK DIFFERENCE WORD

;
; RLMP GET STATUS BIT ASSIGNMENTS
;

WDE=    100000                      ;WRITE DATA ERROR
CHE=    040000                      ;CURRENT HEAD ERROR
WLS=    020000                      ;WRITE LOCK STATUS
SKTO=   010000                      ;SEEK TIMEOUT ERROR
SPD=    004000                      ;SPEED ERROR
WGE=    002000                      ;WRITE GATE ERROR
VC=     001000                      ;VOLUME CHECK
DSE=    000400                      ;DRIVE SELECT ERROR
DT=     000200                      ;DRIVE TYPE
HSS=    000100                      ;HEAD SELECT STATUS
CO=     000040                      ;COVER OPEN
HH=     000020                      ;HEADS HOME
BH=     000010                      ;BRUSHES HOME
SLM=    000005                      ;DRIVE IN SEEK-LINEAR MODE STATE

;
; LOCAL DATA
;
; CONTROLLER IMPURE DATA TABLES
; THESE ARE INDEXED BY THE CONTROLLER NUMBER
;

RTTBL:  .BLKW   R$$L11              ;RETRY COUNT FOR CURRENT OPERATION
PRMSV:  .BLKW   R$$L11*5            ;PARAMETER SAVE AREA FOR WRITE CHECK

;
;DRIVER DISPATCH TABLE
;

        DDTS    DL,R$$L11,NEW=Y ;GENERATE DISPATCH TABLE

;+
; **-DLINI-RL11-RL01/02 DISK CONTROLLER INITIATOR
;
; THIS ROUTINE IS ENTERED FROM THE QUEUE I/O DIRECTIVE WHEN AN I/O
; REQUEST IS QUEUED AND AT THE END OF A PREVIOUS I/O OPERATION TO
; PROPAGATE THE EXECUTION OF THE DRIVER.  IF THE SPECIFIED CONTROLLER
; IS NOT BUSY, THEN AN ATTEMPT IS MADE TO DEQUEUE THE NEXT I/O REQUEST.
; ELSE A RETURN TO THE CALLER IS EXECUTED.  IF THE DEQUEUE ATTEMPT
; IS SUCCESSFUL, THEN THE NEXT I/O OPERATION IS INITIATED.  A RETURN
; TO THE CALLER IS THEN EXECUTED.
;
; INPUTS:
;       R5= ADDRESS OF THE UCB OF THE CONTROLLER TO BE INITIATED.
;
; OUTPUTS:
;       IF THE SPECIFIED CONTROLLER IS NOT BUSY AND AN I/O REQUEST IS
;       WAITING TO BE PROCESSED, THEN THE REQUEST IS DEQUEUED AND THE
;       DRIVER INITIATES THE REQUESTED I/O FUNCTION
;-

DLINI:  GTPKT$  DL,R$$L11       ;GET NEXT I/O PACKET TO PROCESS

;
; THE FOLLOWING ARGUMENTS ARE RETURNED BY $GTPKT:
;
;       R1= ADDRESS OF THE I/O REQUEST PACKET
;       R2= PHYSICAL UNIT NUMBER OF THE REQUESTED DRIVE
;       R3= CONTROLLER INDEX
;       R4= ADDRESS OF THE STATUS CONTROL BLOCK
;       R5= ADDRESS OF THE UCB OF THE DRIVE TO BE INITIATED
;
; RL11-RL01/02 DISK CONTROLLER I/O REQUEST PACKET FORMAT:
;
;       WD. 00 -- I/O QUEUE THREAD WORD
;       WD. 01 -- REQUEST PRIORITY, EVENT FLAG NUMBER
;       WD. 02 -- ADDRESS OF THE TCB OF THE REQUESTOR TASK
;       WD. 03 -- POINTER TO 2ND LUN WORD IN REQUESTOR TASK HEADER
;       WD. 04 -- CONTENTS OF FIRST LUN WORD
;       WD. 05 -- I/O FUNCTION CODE
;       WD. 06 -- VIRTUAL ADDRESS OF I/O STATUS BLOCK
;       WD. 07 -- RELOCATION BIAS OF I/O STATUS BLOCK
;       WD. 10 -- I/O STATUS BLOCK ADDRESS (DISPLACEMENT + 140000)
;       WD. 11 -- VIRTUAL ADDRESS OF AST SERVICE ROUTINE
;       WD. 12 -- MEMORY EXTENSION BITS OF I/O TRANSFER
;       WD. 13 -- BUFFER ADDRESS OF I/O TRANSFER
;       WD. 14 -- NUMBER OF BYTES TO BE TRANSFERRED
;       WD. 15 -- NOT USED.
;       WD. 16 -- LOW BYTE MUST BE ZERO AND HIGH BYTE IS NOT USED
;       WD. 17 -- LOW PART OF LOGICAL BLOCK NUMBER OF I/O REQUEST
;       WD. 20 -- RELOCATION BIAS OF REGISTER BUFFER ELSE NOT USED
;       WD. 21 -- REGISTER BUFFER ADDRESS (DISPLACEMENT + 140000) ELSE NOT USED
;
; DRIVER USAGE OF WORDS IN I/O PACKET:
;       I.PRM+6  (WD. 15) - SEEK DIFFERENCE WORD
;       I.PRM+10 (WD. 16) - STARTING DISK ADDRESS FOR THIS TRANSFER
;       I.PRM+12 (WD. 17) - BYTE COUNT FOR THIS TRANSFER
;

        MOV     #RETRY&377,RTTBL(R3)  ;SET INITIAL RETRY COUNT
        CALL    $VOLVD          ;VALIDATE VOLUME VALID
        BCS     5$              ;IF CS WE FAILED
        TST     R0              ;TRANSFER FUNCTION?
        BMI     10$             ;IF MI YES
        TST     I.PRM+2(R1)     ;SIZE THE DISK?
        BPL     5$              ;IF PL NO, ERROR
        MOV     S.CSR(R4),R2    ;RETRIEVE CSR ADDRESS
        CALL    DLRST           ;RESET DRIVE AND GET STATUS
        MOV     S.PKT(R4),R3    ;RETRIEVE I/O PACKET ADDRESS
        MOV     I.PRM+14(R3),KISAR6   ;SET BUFFER RELOCATION BIAS
        MOV     I.PRM+16(R3),R3 ;GET REGISTER BUFFER ADDRESS
        CALL    REGPAS          ;MOVE REGISTERS INTO BUFFER
5$:     JMP     DLFIN           ;FINISH UP

10$:    CALL    $STMAP          ;SET UP UNIBUS MAPPING ADDRESS
        MOVB    R2,U.BUF+1(R5)  ;SET CURRENT UNIT NUMBER
        MOV     #IE.IFC&377,R0  ;ASSUME ILLEGAL FUNCTION
        BIS     #READ!IE,U.BUF(R5)  ;ASSUME READ LOGICAL FUNCTION
        CMPB    #IO.RLB/256.,I.FCN+1(R1)  ;REALLY?
        BEQ     20$             ;IF EQ YES
        CMPB    #IO.WLB/256.,I.FCN+1(R1)  ;WRITE LOGICAL FUNCTION?
        BNE     5$              ;IF NE NO, EXIT WITH ERROR
        SUB     #WRITE,U.BUF(R5)  ;CONVERT TO WRITE LOGICAL FUNCTION
20$:    MOV     #RETRY,RTTBL(R3)  ;SET INITIAL RETRY COUNT
```

```
        MOV     I.PRM+12(R1),R0 ;RETRIEVE BLOCK NUMBER
        CLR     R2              ;CLEAR HIGH ORDER BLOCK NUMBER
        BITB    #IO.WPB&377,I.FCN(R1)  ;PHYSICAL BLOCK FUNCTION?
        BNE     35$             ;IF NE YES
        CALL    $BLKCK          ;CHECK LOGICAL BLOCK NUMBER
        CMPB    #IO.WLB/256.,I.FCN+1(R3)  ;WRITE FUNCTION?
        BNE     30$             ;IF NE NO
        BITB    #IO.WLT&377,I.FCN(R3)  ;OK TO WRITE ON LAST TRACK?
        BNE     30$             ;IF NE YES
        MOV     R0,I.PRM+6(R3)  ;YES, SAVE STARTING BLOCK NUMBER
        ADD     #^D<20>,I.PRM+12(R3)  ;ADD 1 TRACK'S WORTH OF BLOCKS
        CALL    $BLKC1          ;CHECK IF WRITE ON LAST TRACK OF DISK
        MOV     I.PRM+6(R3),R0  ;RESTORE ORIGINAL STARTING BLOCK NUMBER
30$:    ASL     R0              ;CONVERT BLOCKS TO SECTORS
35$:    MOV     S.PKT(R4),R3    ;RESET I/O PACKET ADDRESS
        CALL    $CVLBN          ;CONVERT BLOCK NUMBER TO DISK ADDRESS
        ROR     R1              ;PUT SURFACE BIT IN CARRY
        ROL     R2              ;MERGE IT WITH THE CYLIDER NUMBER
        ASH     #6,R2           ;POSITION CYLINDER AND SURFACE
        BIS     R0,R2           ;MERGE SECTOR WITH CYLINDER AND SURFACE
        MOV     R2,I.PRM+10(R3) ;SAVE STARTING DISK ADDRESS
        MOV     U.CNT(R5),I.PRM+12(R3)  ;ASSUME ONLY ONE XFER NEEDED
        MOV     #^D<40>,R1      ;SET SECTORS/SURFACE
        SUB     R0,R1           ;CALCULATE SECTORS LEFT ON SURFACE
        SWAB    R1              ;GET BYTES LEFT ON SURFACE
        CMP     U.CNT(R5),R1    ;ARE ADDITIONAL TRANSFERS REQUIRED?
        BLOS    40$             ;IF LOS NO
        MOV     R1,I.PRM+12(R3) ;SET BYTE COUNT FOR FIRST TRANSFER
40$:    MOVB    S.CON(R4),R1    ;RETRIEVE CONTROLLER INDEX
        MUL     #5,R1           ;FORM INDEX INTO PARAMETER SAVE AREA
        ADD     #PRMSV,R1       ;POINT TO THIS ENTRY
        MOV     U.BUF(R5),(R1)+ ;SAVE INITIAL PARAMETERS
        MOV     U.BUF+2(R5),(R1)+  ;...
        MOV     U.CNT(R5),(R1)+ ;...
        MOV     I.PRM+10(R3),(R1)+  ;...
        MOV     I.PRM+12(R3),(R1)+  ;...

;+
; THIS SECTION WILL INITIATE THE OPERATION
;-

DLINIO: CALL    $MPUBM          ;MAP UNIBUS TO TRANSFER
        MOV     S.CSR(R4),R2    ;GET ADDRESS OF CSR
        MOV     S.PKT(R4),R3    ;GET ADDRESS OF I/O PACKET
        CLRB    U.CW2+1(R5)     ;RESET DRIVE SETTLE DOWN FLAG
        CLR     I.PRM+6(R3)     ;RESET ERROR DIFFERENCE WORD
        MOVB    S.ITM(R4),S.CTM(R4)  ;SET DEVICE TIMEOUT COUNTER
        CALL    DLRST           ;RESET DRIVE AND GET STATUS
        MOV     RLMP(R2),R1     ;GET THE STATUS INFO
        BIC     #WLS!DT!HSS,R1  ;REMOVE IRRELEVANT BITS
        BIT     #DRDY,(R2)      ;IS THE DRIVE READY?
        BEQ     10$             ;IF EQ NO
        CMP     #HH!BH!SLM,R1   ;HEADS, BRUSHES AND STATE OK?
        BEQ     20$             ;IF EQ YES
10$:    BITB    #US.SPU,U.STS(R5)  ;IS DRIVE SPINNING UP?
        BNE     DLPWF1          ;IF NE YES, WAIT FOR IT TO SPIN UP
        MOV     #IE.DNR&377,R0  ;SET RETURN ERROR CODE
        JMP     DLFIN           ;EXIT WITH FATAL ERROR
20$:    BICB    #US.SPU,U.STS(R5)  ;RESET DRIVE SPINNING UP
        MOV     I.PRM+10(R3),R0 ;RETRIEVE STARTING DISK ADDRESS
        CALL    DLDIFF          ;CALCULATE DIFFERENCE WORD
DLGO:   BEQ     30$             ;IF EQ NO SEEK IS NECESSARY
        MOV     #SEEK,R1        ;GET CODE FOR SEEK FUNCTION
        CALL    DLXCT           ;EXECUTE THE SEEK
        BMI     DLEROR          ;IF MI ERROR DURING SEEK FUNCTION
30$:    ADD     #RLMP,R2        ;POINT TO RLMP
```

```
        MOV     I.PRM+12(R3),R1 ;GET BYTE COUNT
        ROR     R1              ;MAKE IT A WORD COUNT
        NEG     R1              ;ALSO NEGATIVE
        MOV     R1,(R2)         ;LOAD WORD COUNT
        MOV     I.PRM+10(R3),-(R2)  ;LOAD STARTING DISK ADDRESS
        MOV     U.BUF+2(R5),-(R2)   ;LOAD BUS ADDRESS
        CALL    $BMSET          ;SET I/O ACTIVE BIT IN MAP
        MOV     U.BUF(R5),-(R2) ;;;LOAD FUNCTION AND GO

;+
; CANCEL I/O OPERATION IS A NOP FOR FILE STRUCTURED DEVICES.
;-

DLCAN:  RETURN                  ;;;NOP FOR RL11

;+
; POWERFAIL IS HANDLED VIA THE DEVICE TIMEOUT FACILITY AND
; CAUSES NO IMMEDIATE ACTION ON THE UNIT.  THE CURRENT TIMEOUT
; COUNT IS EXTENDED, THUS IF A UNIT WAS BUSY IT WILL HAVE
; SUFFICIENT TIME TO SPIN BACK UP.  THE NEXT I/O REQUEST TO ANY
; UNIT WILL BE SUSPENDED FOR AT LEAST THE EXTENDED TIMEOUT UNLESS
; THE UNIT IS CURRENTLY READY.
;-

DLPWF:  TSTB    S.STS(R4)       ;IS DRIVE CURRENTLY BUSY?
        BEQ     DLPWF2          ;IF EQ NO
        MOVB    #4,S.STS(R4)    ;ALLOW FOR A FULL MINUTE TO SPIN UP
DLPWF1: MOVB    #RLSPU,S.CTM(R4)  ;EXTEND TIMEOUT INCASE UNIT WAS BUSY
DLPWF2: BISB    #US.SPU,U.STS(R5)  ;SET UNIT SPINNING UP
        RETURN                  ;

;+
; **-$DLINT-RL11-RL01/02 DISK CONTROLLER
;       INTERRUPT AND ERROR SERVICE ROUTINES
;-
        .ENABL  LSB

$DLINT::INTSV$  DL,PR5,R$$L11   ;;;SAVE REGISTERS AND SET PRIORITY
        CALL    $FORK           ;;;CREATE A SYSTEM PROCESS
        MOV     R4,R3           ;COPY CONTROLLER INDEX
        ASRB    RTTBL+1(R3)     ;HOME SEEK IN PROGRESS?
        BCS     DLINIO          ;IF CS YES
        MOV     U.SCB(R5),R4    ;GET ADDRESS OF SCB
        MOV     S.CSR(R4),R2    ;GET ADDRESS OF CSR
        MOV     #IS.SUC&377,R0  ;ASSUME SUCCESSFUL OPERATION
        MOV     S.PKT(R4),R3    ;RETRIEVE I/O PACKET ADDRESS
        MOV     (R2),R1         ;GET CONTENTS OF RLCS
        BMI     20$             ;IF MI AN ERROR OCCURRED
        SUB     I.PRM+12(R3),U.CNT(R5)  ;CALCULATE BYTES LEFT TO XFER
        BEQ     70$             ;IF EQ NONE LEFT
        MOV     U.CNT(R5),I.PRM+12(R3)  ;ASSUME LAST XFER COMING
        CMP     U.CNT(R5),#RLBPT  ;IS THIS THE LAST TRANSFER?
        BLOS    10$             ;IF LOS YES
        MOV     #RLBPT,I.PRM+12(R3)  ;TRANSFER A WHOLE TRACKS WORTH
10$:    BIC     #CRDY,R1        ;CLEAR CRDY TO START FUNCTION
        MOV     R1,U.BUF(R5)    ;SAVE CURRENT FUNCTION AND ADDRESS BITS
        MOV     RLBA(R2),U.BUF+2(R5)  ;SAVE CURRENT BUS ADDRESS
        MOV     I.PRM+10(R3),R0 ;GET INITIAL DISK ADDRESS
        MOV     R0,R1           ;COPY DISK ADDRESS
        BIS     #77,R0          ;UPDATE CYLINDER AND SURFACE ...
        INC     R0              ;... LEAVING SECTOR BITS ZERO
        MOV     R0,I.PRM+10(R3) ;SAVE NEW DISK ADDRESS
        CALL    DLDIFO          ;CALCULATE MID-TRANSFER DIFFERENCE
        JMP     DLGO            ;GO DO THE OPERATION

20$:    BIT     #DRDY,R1        ;IS THE DRIVE READY?
```

```
        BNE     DLEROR              ;IF NE YES, GO CHECK FOR ERRORS
25$:    MOVB    #3,S.CTM(R4)        ;WAIT 3 SECONDS FOR THE DRIVE TO SETTLE
        INCB    U.CW2+1(R5)         ;FLAG SETTLE DOWN IN PROGRESS
        RETURN                      ;

DLEROR: MOV     (R2),R1             ;RETRIEVE CONTENTS OF RLCS
        MOV     #IE.VER&377,R0      ;ASSUME UNRECOVERABLE ERROR
        BIT     #NXM,R1             ;NON-EXISTENT MEMORY?
        BNE     90$                 ;IF NE YES
        BIT     #DE,R1              ;DRIVE PROBLEMS?
        BEQ     40$                 ;IF EQ NO
        CALL    DLGST               ;EXECUTE GET DRIVE STATUS FUNCTION
        BIT     #WGE,RLMP(R2)       ;WRITE GATE ERROR?
        BEQ     90$                 ;IF EQ NO
        BIT     #WLS,RLMP(R2)       ;IS THE DRIVE WRITE LOCKED?
        BEQ     DLRTRY              ;IF EQ NO
        MOV     #IE.WLK&377,R0      ;SET WRITE LOCK ERROR CODE
        BR      DLFIN               ;
40$:    BIT     #10,U.BUF(R5)       ;WRITE CHECK FUNCTION?
        BNE     DLRTRY              ;IF NE NO
        BIT     #OPI,R1             ;OPERATION INCOMPLETE?
        BNE     DLRTRY              ;IF NE YES
        BIT     #DCK,R1             ;WRITE CHECK ERROR?
        BEQ     DLRTRY              ;IF EQ NO
        MOV     #IE.WCK&377,R0      ;YES, SET WRITE CHECK ERROR CODE
        BR      DLRTRY              ;GO RETRY OPERATION IF REQUIRED

70$:    BITB    #IO.WLC&377,I.FCN(R3)  ;WRITE WITH WRITE CHECK?
        BNE     80$                 ;IF NE YES
        BITB    #US.WCK,U.STS(R5)   ;WRITE CHECK ENABLED?
        BEQ     DLFIN               ;IF EQ NO
80$:    MOV     U.BUF(R5),R1        ;GET CURRENT FUNCTION CODE
        BIT     #WCHK,R1            ;WRITE OR WRITE CHECK FUNCTION?
        BEQ     DLFIN               ;IF EQ NO
        BIT     #10,R1              ;WAS FUNCTION WRITE CHECK?
        BEQ     DLFIN               ;IF EQ YES
        MOVB    S.CON(R4),R1        ;RETRIEVE CONTROLLER INDEX
        MOV     #RETRY,RTTBL(R1);RESET RETRY COUNT
        MUL     #5,R1               ;FORM AN INDEX INTO SAVE AREA
        ADD     #PRMSV,R1           ;...
        MOV     (R1)+,U.BUF(R5) ;RESTORE STARTING PARAMETERS
        MOV     (R1)+,U.BUF+2(R5)   ;...
        MOV     (R1)+,U.CNT(R5) ;...
        MOV     (R1)+,I.PRM+10(R3)  ;...
        MOV     (R1)+,I.PRM+12(R3)  ;...
        BIC     #10,U.BUF(R5)       ;CONVERT TO WRITE CHECK FUNCTION
        JMP     DLINIO              ;START THE WRITE CHECK

;+
; FINISH I/O OPERATION
;-

90$:    MOV     #IE.VER&377,R0      ;SET UNSUCCESSFUL OPERATION
DLFIN:  MOV     S.PKT(R4),R2        ;GET ADDRESS OF I/O PACKET
        MOV     I.PRM+4(R2),R1      ;GET TOTAL TRANSFER SIZE
        SUB     U.CNT(R5),R1        ;CALCULATE BYTES TRANSFERRED
        MOVB    S.CON(R4),R3        ;RETRIEVE CONTROLLER INDEX
        MOVB    RTTBL(R3),R2        ;GET FINAL RETRY COUNT
        BIS     #RETRY*^D<256>,R2   ;MERGE STARTING RETRY COUNT
        CALL    $IODON              ;FINISH I/O OPERATION
        JMP     DLINI               ;PROCESS NEXT REQUEST

;+
; **-DLOUT-RL11-RL01/02 DISK CONTROLLER
;       DEVICE TIMEOUT ROUTINE
;
```

```
; DEVICE TIMEOUT RESULTS IN THE OPERATION BEING REPEATED.
; TIMEOUTS ARE USUALLY CAUSED BY A POWER FAILURE BUT MAY ALSO
; BE THE RESULT OF A HARDWARE MALFUNCTION.
;-

DLOUT:  MOV     S.PKT(R4),R3      ;;;RETRIEVE I/O PACKET ADDRESS
        BITB    #US.SPU,U.STS(R5)  ;;;IS DRIVE SPINNING UP?
        BEQ     20$               ;;;IF EQ NO
        DECB    S.STS(R4)         ;;;HAVE WE WAITED A MINUTE YET?
        BNE     10$               ;IF NE NO
        INCB    S.STS(R4)         ;;;LEAVE CONTROLLER BUSY
        BR      30$               ;;;LOG DEVICE TIMEOUT
10$:    MTPS    #0                ;;;ALLOW INTERRUPTS
        JMP     DLINIO            ;RETRY ENTIRE OPERATION
20$:    TSTB    U.CW2+1(R5)       ;;;IS DRIVE SETTLING DOWN?
        BEQ     30$               ;;;IF EQ NO
        MTPS    #0                ;;;YES, ALLOW INTERRUPTS
        JMP     DLEROR            ;PROCESS THE ERROR
30$:    MTPS    #0                ;;;ALLOW INTERRUPTS
        CALL    DLRST             ;RESET DRIVE
        MOV     #IE.DNR&377,R0    ;SET DEVICE NOT READY
DLRTRY: MOV     S.PKT(R4),R1      ;GET I/O PACKET ADDRESS
        BITB    #IQ.X,I.FCN(R1)   ;INHIBIT RETRIES?
        BNE     DLFIN             ;IF NE YES
        DECB    RTTBL(R3)         ;ANY MORE RETRIES LEFT?
        BLE     DLFIN             ;IF LE NO
        JMP     DLINIO            ;YES, RETRY ENTIRE OPERATION

;+
; **-DLXCT,DLGST,DLRST-RL11-RL01/02 DISK CONTROLLER
;       FUNCTION EXECUTION ROUTINES
;
; THIS ROUTINE WILL EXECUTE A GET DRIVE STATUS OR ANY
; NON-INTERRUPTABLE FUNCTION AND WAIT FOR ITS COMPLETION.
;
; INPUTS:
;       R1 = FUNCTION CODE
;       R2 = CSR ADDRESS
;       R5 = UCB ADDRESS
;
; OUTPUTS:
;       R1 = CONTENTS OF RLCS (TESTED)
;       FUNCTION EXECUTED
;-

        .ENABL  LSB
DLRST:  MOV     #RST!STS!MRK,RLDA(R2)  ;SET MESSAGE CODES IN RLDA
        CALL    10$               ;DO THE DRIVE RESET FIRST
DLGST:  MOV     #STS!MRK,RLDA(R2)  ;SET MESSAGE CODES IN RLDA
10$:    MOV     #GSTS,R1          ;SET GET STATUS FUNCTION
DLXCT:  MOV     R1,-(SP)          ;SAVE FUNCTION CODE
        MOVB    U.UNIT(R5),1(SP)  ;MERGE CURRENT DRIVE BITS
        MOV     (SP)+,(R2)        ;LOAD RLCS
20$:    BIT     #ERR!CRDY,(R2)    ;READY OR ERROR?
        BEQ     20$               ;IF EQ NEITHER
        MOV     (R2),R1           ;SAVE RLCS AND TEST FOR ERRORS
        RETURN                    ;
        .DSABL  LSB
;+
; **-DLDIFF-RL11-RL01/02 DISK CONTROLLER
;       CYLINDER ADDRESS DIFFERENCE CALCULATOR
;
; THIS SUBROUTINE CALCULATES THE DIFFERENCE WORD USED IN THE
; SEEK OPERATION.  IF A HEADER CANNOT BE READ AFTER 16. RETRIES,
; AN ERROR WILL BE LOGGED AND A ONE CYLINDER REVERSE SEEK WILL BE
; ISSUED.  THE SEEK IS FOLLOWED BY A READ HEADERS TO CAUSE AN
; INTERRUPT.
;
; INPUTS:
;       R0 = DESIRED DISK ADDRESS
;       R3 = I/O PACKET ADDRESS
;
; OUTPUTS:
;       R1 = DIFFERENCE WORD
;       RLDA = LOADED WITH DIFFERENCE WORD
;       I.PRM+6 = LOADED WITH DIFFERENCE WORD
;       IF EQ NO SEEK IS NECESSARY
;-

DLDIFF: MOV     #RETRY*2,-(SP)  ;SET READ HEADER RETRY COUNT
10$:    MOV     #RDH,R1         ;SET CODE FOR READ HEADERS FUNCTION
        CALL    DLXCT           ;EXECUTE THE FUNCTION
        BPL     20$             ;IF PL FUNCTION EXECUTED OK
        DEC     (SP)            ;ANY RETRIES LEFT?
        BGT     10$             ;IF GT YES
        CMP     (SP)+,(SP)+     ;REMOVE RETRY COUNT AND CALLERS ADDRESS
        CALL    DLRST           ;RESET DRIVE
        MOV     #REV,RLDA(R2)   ;LOAD REVERSE SEEK DIFFERENCE WORD
        MOV     #SEEK,R1        ;GET CODE FOR SEEK FUNCTION
        MOV     #IE.VER&377,R0  ;ASSUME WE WILL FAIL
        CALL    DLXCT           ;EXECUTE THE SEEK
        BMI     DLFIN           ;IF MI WE FAILED
        BIC     #377,R1         ;CLEAR OUT FUNCTION BITS
        BIS     #IE!RDH,R1      ;LOAD CODES FOR READ HEADER
        MOVB    #1,RTTBL+1(R3)  ;INDICATE REVERSE SEEK IN PROGRESS
        MOVB    S.ITM(R4),S.CTM(R4)  ;SET DEVICE TIMEOUT COUNTER
        MOV     R1,(R2)         ;LOAD FUNCTION AND GO
        RETURN                  ;WAIT FOR THE INTERRUPT
20$:    TST     (SP)+           ;REMOVE RETRY COUNT
        MOV     RLMP(R2),R1     ;RETRIEVE HEADER WORD
DLDIF0: CLR     I.PRM+6(R3)     ;RESET DIFFERENCE WORD
        BIC     #77,R0          ;MASK OUT SECTOR BITS
        BIC     #77,R1          ;...
        CMP     R0,R1           ;DO WE NEED TO DO A SEEK?
        BEQ     40$             ;IF EQ NO
        MOV     R0,-(SP)        ;SAVE DESIRED DISK ADDRESS
        BIC     #^C<100>,(SP)   ;ISOLATE SURFACE BIT
        ASR     (SP)            ;PUT INTO THE PROPER POSITION
        ASR     (SP)            ;...
        BIC     #100,R0         ;REMOVE SURFACE BIT
        BIC     #100,R1         ;...
        SUB     R0,R1           ;SUBTRACT DESIRED FROM ACTUAL
        BCC     30$             ;IF CC ACTUAL >= DESIRED
        NEG     R1              ;ACTUAL < DESIRED, MAKE POSITIVE DIFFERENCE
        BIS     #SN,R1          ;SET SIGN FOR MOVE TO CENTER OF DISK
30$:    INC     R1              ;SET MARKER BIT
        BIS     (SP)+,R1        ;MERGE IN SURFACE BIT
        MOV     R1,RLDA(R2)     ;LOAD DIFFERENCE WORD
        MOV     R1,I.PRM+6(R3)  ;SAVE DIFFERENCE WORD
40$:    RETURN                  ;


;+
; MOVE THE CONTROLLER/DRIVE REGISTERS INTO THE SPECIFIED BUFFER.
;
; INPUTS:
;       R2 = CSR ADDRESS
;       R3 = BUFFER ADDRESS
;-

REGPAS: MOV     (R2),(R3)+      ;MOVE RLCS
        MOV     RLBA(R2),(R3)+  ;MOVE RLBA
        MOV     RLDA(R2),(R3)+  ;MOVE RLDA
        MOV     RLMP(R2),(R3)+  ;MOVE RLMP
        CLR     (R3)+           ;CLEAR PLACE HOLDERS...
        CLR     (R3)+           ;...SO HRC/CON WILL WORK
        CALL    DLGST           ;EXECUTE GET DRIVE STATUS FUNCTION
        MOV     RLMP(R2),(R3)   ;SAVE DRIVE STATUS
        RETURN                  ;

;+
; **-DLKRB-CONTROLLER ON-LINE/OFF-LINE ROUTINE
;
; THIS ROUTINE WILL HANDLE RECONFIGURATION CALLS FOR ON-LINE
; CONTROLLER AND OFF-LINE CONTROLLER FOR THE RL11.
;
; INPUTS:
;       R2 = KRB ADDRESS
;       R3 = CTB ADDRESS
;       C=1 IF OFF-LINE REQUEST
;       C=0 IF ON-LINE REQUEST
;
; OUTPUTS:
;       NONE
;-

DLKRB:  BCS     DLOFL           ;HANDLE OFF-LINE REQUEST

;
; CODE SPECIFIC TO HANDLE THE CONTROLLER COMING ON-LINE.
;

        RETURN                  ;EXIT

;
; CODE SPECIFIC TO HANDLE THE CONTROLLER GOING OFF-LINE
;

DLOFL:                          ;
        RETURN                  ;

;+
; **-DLUCB-UNIT ON-LINE/OFF-LINE ROUTINES
;
; THIS ROUTINE WILL HANDLE RECONFIGURATION CALLS FOR ON-LINE
; UNIT AND OFF-LINE UNIT FOR RL01 AND RL02 DRIVES.
;
; INPUTS:
;       R3 = CONTROLLER INDEX
;       R4 = SCB ADDRESS
;       R5 = UCB ADDRESS
;       C=1 IF OFF-LINE REQUEST
;       C=0 IF ON-LINE REQUEST
;
; OUTPUTS:
;       NONE
;-

DLUCB:  BCS     DLOFLU          ;IF CS OFF-LINE REQUEST

;
; CODE SPECIFIC TO BRINGING UNIT ON-LINE.
;

        RETURN                  ;

;
; CODE SPECIFIC TO TAKING UNIT OFF-LINE.
```

```
;

DLOFLU:                         ;
        RETURN                  ;


        .END
```

8.3 HANDLING SPECIAL USER BUFFERS

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

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

The coding shown in this section 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:

   a. Retrieves the user virtual address and address-checks it

   b. Relocates the virtual address and stores the result back into the packet

   c. 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
;

        $BMTBL=0                        ;LOADABLE BMDRV

$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   $BMTBL          ; 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

BMST=.

        .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=.

;
;                                                               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:
;
;       ...     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 WAIT-
;       ING TO BE PROCESSED, THEN THE REQUEST IS DEQUEUED AND THE I/O OPER-
;       ATION 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 ADDRES (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 ISCURRENTLY 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

;       +----------------------------------------------------------------+
;       |                                                                |
;       |       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 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 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
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
```

```
;       ****************************************************************
;       *                                                              *
;       *       A REGION AST WAS QUEUED. BUMP BUFFERED I/O COUNT       *
;       *       BACK UP TO FORCE I/O RUNDOWN IN CASE OF ABORT AND      *
;       *       EXIT AST SERVICE ROUTINE.                              *
;       *                                                              *
;       ****************************************************************

        MOV     I.TCB(R5),R0    ; GET TCB ADDRESS
        INCB    T.TIO(R0)       ; BUMP BUFFERED I/O COUNT
        RETURN                  ; EXIT AST SERVICE ROUTINE
;       ****************************************************************
;       *                                                              *
;       *       PERFORM BUFFER COPY OPERATION                          *
;       *                                                              *
;       ****************************************************************

60$:    MOV     I.TCB(R5),R0    ; GET TCB ADDRESS OF TASK
        INCB    T.IOC(R0)       ; ADJUST REAL I/O COUNT UPWARDS
        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          ; RE-QUEUE 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
```

APPENDIX A

RSX-11M-PLUS SYSTEM DATA STRUCTURES AND SYMBOLIC DEFINITIONS

This appendix describes the RSX-11M-PLUS 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$
     DCBDF$                   ;Define DCB offsets
```

NOTE

> All physical offsets and bit definitions are subject to change in future releases of the operating system. Code that accesses system data structures should always use the symbolic offsets rather than the physical offsets.

The first two arguments, <:> and <=>, make all definitions global. If they are left blank, the definitions will be local. The SYSDEF argument causes the variable part of a data structure to be defined.

All of these macros are in the Executive macro library, LB:[1,1]EXEMC.MLB. All except F11DF$, ITBDF$, MTADF$, OLRDF$, and SHDDF$ 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$	<:>,<=>	Accounting data structures (user account block, task account block, system account block)
CLKDF$	<:>,<=>	Clock queue control block

(continued on next page)

Table A-1 (Cont.)
Summary of System Data Structure Macros

Macro	Arguments	Data Structures
CTBDF$	<:>,<=>	Controller table
DCBDF$	<:>,<=>,SYSDEF	Device control block
EPKDF$	<:>,<=>	Error message block
F11DF$	<:>,<=>,SYSDEF	Files-11 data structures (volume control block, mount list entry, file control block, file window block, locked block list node)
HDRDF$	<:>,<=>	Task header and window block
HWDDF$	<:>,<=>,SYSDEF	Hardware register addresses and feature mask definitions
ITBDF$	<:>,<=>,SYSDEF	Interrupt transfer block
KRBDF$	<:>,<=>	Controller request block
LCBDF$	<:>,<=>	Logical assignment control block
MTADF$	<:>,<=>	ANSI magtape data structures (volume set control block)
OLRDF$		On-line reconfiguration interface
PCBDF$	<:>,<=>,SYSDEF	Partition control block and attachment descriptor
PKTDF$	<:>,<=>	I/O packet, AST control block, offspring control block, group global event flag control block, and CLI parser block
SCBDF$	<:>,<=>,SYSDEF	Status control block and UMR assignment block
SHDDF$	<:>,<=>	Shadow recording linkage block
TCBDF$	<:>,<=>,SYSDEF	Task control block
UCBDF$	<:>,<=>,TTDEF,SYSDEF	Unit control block

ABODF$

```
        ABODF$

;
; TASK ABORT CODES
;
; NOTE: S.COAD-S.CFLT ARE ALSO SST VECTOR OFFSETS
;
S.CACT=-4.              ;TASK STILL ACTIVE
S.CEXT=-2.              ;TASK EXITED NORMALLY
S.COAD=0.               ;ODD ADDRESS AND TRAPS TO 4
S.CSGF=2.               ;SEGMENT FAULT
S.CBPT=4.               ;BREAK POINT OR TRACE TRAP
S.CIOT=6.               ;IOT INSTRUCTION
S.CILI=8.               ;ILLEGAL OR RESERVED INSTRUCTION
S.CEMT=10.              ;NON RSX EMT INSTRUCTION
S.CTRP=12.              ;TRAP INSTRUCTION
S.CFLT=14.              ;11/40 FLOATING POINT EXCEPTION
S.CSST=16.              ;SST ABORT-BAD STACK
S.CAST=18.              ;AST ABORT-BAD STACK
S.CABO=20.              ;ABORT VIA DIRECTIVE
S.CLRF=22.              ;TASK LOAD REQUEST FAILURE
S.CCRF=24.              ;TASK CHECKPOINT READ FAILURE
S.IOMG=26.              ;TASK EXIT WITH OUTSTANDING I/O
S.PRTY=28.              ;TASK MEMORY PARITY ERROR
S.CPMD=30.              ;TASK ABORTED WITH PMD REQUEST
S.CELV=32.              ;TI: VIRTUAL TERMINAL WAS ELIMINATED
S.CINS=34.              ;TASK INSTALLED IN 2 DIFFERENT SYSTEMS
S.CAFF=36.              ;TASK ABORTED DUE TO BAD AFFINITY (REQUIRED
                        ;BUS RUNS ARE OFFLINE OR NOT PRESENT)
S.CCSM=38.              ;BAD CSM PARAMETERS OR BAD STACK
S.COTL=40.              ;TASK HAS RUN OVER ITS TIME LIMIT


;
; TASK TERMINATION NOTIFICATION MESSAGE CODES
;
T.NDNR=0                ;DEVICE NOT READY
T.NDSE=2                ;DEVICE SELECT ERROR
T.NCWF=4                ;CHECKPOINT WRITE FAILURE
T.NCRE=6                ;CARD READER HARDWARE ERROR
T.NDMO=8.               ;DISMOUNT COMPLETE
T.NUER=10.              ;UNRECOVERABLE ERROR
T.NLDN=12.              ;LINK DOWN (NETWORKS)
T.NLUP=14.              ;LINK UP (NETWORKS)
T.NCFI=16.              ;CHECKPOINT FILE INACTIVE
T.NUDE=18.              ;UNRECOVERABLE DEVICE ERROR
T.NMPE=20.              ;MEMORY PARITY ERROR
T.NKLF=22.              ;UCODE LOADER NOT INSTALLED
T.NAAF=24.              ;ACCOUNTING ALLOCATION FAILURE
T.NTAF=26.              ;ACCOUTING TAB ALLOCATION FAILURE
T.NDEB=28.              ;TASK HAS NO DEBUGGING AID
T.NRCT=30.              ;REPLACEMENT CONTROL TASK NOT INSTALLED
T.NWBL=32.              ;WRITE BACK CACHING DATA LOST
                        ;UNIT WRITE LOCKED
```

ACNDF$

```
                       ACNDF$

        ;
        ; ACCOUNTING  BLOCK OFFSET AND STATUS DEFINITIONS
        ; FOR EACH TRANSACTION TYPE.
        ;
        ;
        ; HEADER COMMON TO ALL TRANSACTIONS
        ;
                       .ASECT
        .=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-RAD50, SECOND-BINARY
000016  B.ACN:  .BLKW   1       ;ACCOUNT NUMBER
000020  B.TID:  .BLKB   1       ;ASCII TERMINAL TYPE (V,T,B 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 BOUNDRY
        ;
        .=$$$TLN                ;START AFTER TOTALS
000065  B.USE:  .BLKB   1       ;USE COUNT
000066  B.ACT:  .BLKW   1       ;NUMBER OF CURRENTLY ACTIVE TASKS
000070  B.UUIC: .BLKW   1       ;LOGIN UIC
000072  B.UCB:  .BLKW   1       ;POINTER TO UCB
000074  B.LGO:  .BLKW   3       ;LOGOFF TIME
000102  B.ULNK: .BLKW   1       ;LINK TO NEXT UAB
000104  B.RNA:  .BLKW   3       ;LOC IN SYSTEM ACCNT FILE
                                ;(OFFSET,VBN-HI,VBN-LO)
000112  B.NAM:  .BLKB   14.     ;LAST NAME OF USER
000130          .BLKB   1       ;FIRST INITIAL OF USER
000131          .BLKB   1       ;UNUSED BYTE
000132  B.ULEN=.                ;UAB LENGTH
000002  $$$=<.+77>/100          ;UAB LENGTH (ROUNDED UP TO 32 WORD BOUND)
```

ACNDF$ (Cont.)

```
        ;
        ; TASK ACCOUNT BLOCK (TAB)
        ;       NOTE: THE TAB MUST END ON A WORD BOUNDRY
        ;
        .=$$$TLN                        ;STARTS AFTER TOTALS
000065  B.PRI:  .BLKB   1               ;HIGHEST RUNNING PRIORITY
000066  B.TNAM: .BLKW   2               ;TASK NAME
000072  B.TCB:  .BLKW   1               ;TCB ADDRESS
000074  B.TST3: .BLKW   1               ;T.ST3 FROM TASK'S TCB
000076          .BLKW   1               ;RESERVED FOR FUTURE STATUS BITS
000100  B.CUIC: .BLKW   1               ;CURRENT UIC OF TASK
000102  B.PUIC: .BLKW   1               ;PROTECTION UIC OF TASK
000104  B.CTXT: .BLKW   2               ;NUMBER OF CONTEXT LOADS
000110  B.TCKP: .BLKW   2               ;TIMES TASK HAS BEEN CHECKPOINTED
000114  B.OVLY: .BLKW   2               ;NUMBER OF DISK OVERLAY LOADS
000120  B.EXST: .BLKW   2               ;EXIT STATUS AND ABORT CODE
000124  B.TLEN=.                        ;TAB LENGTH
000002  B.TBLK=<.+77>/100               ;NUMBER OF SEC POOL BLOCKS IN TAB


        ;
        ; SYSTEM ACCOUNT BLOCK (SAB)
        ;
        .=$$$TLN                        ;START AFTER TOTALS
000065  B.SHDN: .BLKB   1               ;ACCOUNTING SHUTDOWN REASON CODE
000066  B.UHD:  .BLKW   1               ;UAB LISTHEAD
000070  B.ULO:  .BLKW   1               ;NUMBER OF USERS CURRENTLY LOGGED ON
000072  B.ULT:  .BLKW   2               ;TOTAL NUMBER OF LOGONS
000076  B.CKP:  .BLKW   2               ;TOTAL NUMBER OF CHECKPOINTS
000102  B.SHF:  .BLKW   2               ;TOTAL NUMBER OF SHUFFLER RUNS
000106  B.RND:  .BLKW   2               ;NUMBER OF CPU INTERVALS ROUNDED UP TO 1
000112  B.FID:  .BLKW   3               ;FILE-ID OF TRANSACTION FILE
000120  B.DVNM: .BLKB   2               ;DEVICE  OF TRANSACTION FILE
000122  B.UNIT: .BLKW   1               ;UNIT    OF TRANSACTION FILE
000124  B.EXTS: .BLKW   1               ;EXTEND SIZE FOR TRANSACTION FILE
000126  B.LSCN: .BLKW   3               ;TIME OF LAST SCAN
000134  B.SCNR: .BLKW   1               ;SCAN RATE IN SECONDS
000136  B.DSCN: .BLKW   1               ;STATISTICAL SCAN RATE (IN SEC)
000140  B.STSP: .BLKW   2               ;RESERVED
000144  B.SYSM: .BLKW   1               ;RESERVED
000146  B.CKUS: .BLKW   3               ;RESERVED
000154  B.CKSP: .BLKW   2               ;RESERVED
000160  B.CKAL: .BLKW   1               ;RESERVED
000162  B.SLEN=.                        ;SAB LENGTH


        ;
        ; NEW FIELDS FOR EXTENDED ACCOUNTING
        ;
000162  B.CPUT: .BLKW   8.              ;CPU TIME USED PER PROCESSOR
000202  B.CTXP: .BLKW   8.              ;NUMBER OF CONTEXT SWITCHES (PER PROC)
000222  B.IDCT: .BLKW   8.              ;NUMBER OF IDLE LOOP ENTRIES (PER PROC)
000242  B.QIOC: .BLKW   8.              ;NUMBER OF I/O INITIATIONS (PER PROC)
000262  B.MIOC: .BLKW   8.              ;MASS STORE I/O COMPLETIONS (PER PROC)
000302  B.AIOC: .BLKW   8.              ;ALL I/O COMPLETIONS (PER PROC)
000322  B.IPSN: .BLKW   8.              ;IP INTERRUPTS SENT (PER PROC)
000342  B.IPRC: .BLKW   8.              ;IP INTERRUPTS RCVD (PER PROC)
000362  B.CKEX: .BLKW   2               ;CHECKPOINT DUE TO EXTEND TASKS
000366  B.CFCL: .BLKW   2               ;CALLS TO CFORK
000372  B.CFRK: .BLKW   2               ;CFORK FORKS
000376  B.TLOD: .BLKW   2               ;TASK LOADS
```

ACNDF$ (Cont.)

```
000402  B.RLOD: .BLKW   2         ;REGION LOADS
000406          .BLKB   82.       ;BUMP SIZE TO NEXT 32 WORD BLOCK
000346  B.SSBL=.-B.SLEN           ;EXTRA LENGTH OF SYSTEM STATISTICS BLOCK
000006  $$$=<.+77>/100            ;SAB LENGTH (ROUNDED UP TO 32 WORD BOUND)


        ;
        ; SYSLOG STARTUP TRANSACTION
        ;
        .=$$$HLN                  ;START AFTER HEADER
000022  B.SSLN=.                  ;TRANSACTION LENGTH


        ;
        ; CRASH RECOVERY TRANSACTION
        ;
        .=$$$HLN                  ;START AFTER STANDARD HEADER
000022  B.CTLS: .BLKW   3         ;TIME OF LAST SCAN BEFORE CRASH
000030  B.CSRT: .BLKW   1         ;SCAN RATE BEFORE CRASH
000032  B.CRSN: .BLKB   60.       ;ASCII TEXT EXPLAINING CRASH
000126  B.CLEN=.                  ;TRANSACTION LENGTH


        ;
        ; INVALID LOGIN TRANSACTION
        ;
        .=$$$HLN                  ;
000022  B.INAM: .BLKB   14.       ;NAME FROM LOGIN LINE
000040  B.IUIC: .BLKB   6.        ;UIC FROM LOGIN LINE
000046  B.IPSW: .BLKB   6.        ;PASSWORD FROM LOGIN LINE
000054  B.ILEN=.                  ;TRANSACTION LENGTH


        ;
        ; DEVICE TRANSACTIONS (ALLOCATION, DEALLOCATION, MOUNT, AND
        ;                      DISMOUNT)
        ;
        .=$$$HLN                  ;
000022  B.DNAM: .BLKW   1         ;ASCII DEVICE NAME
000024  B.DUNT: .BLKB   1         ;OCTAL DEVICE UNIT NUMBER
000025  B.DLEN=.                  ;TRANSACTION LENGTH FOR ALL, DEA, AND DMO
000025          .BLKB   1         ;UNUSED BYTE
000026  B.DLBL: .BLKW   6         ;VOLUME LABEL
000042  B.DMST: .BLKW   1         ;MOUNT STATUS BITS
000044  B.DUIC: .BLKW   1         ;OWNER UIC
000046  B.DVPR: .BLKW   1         ;VOLUME PROTECTION CODE
000050  B.DACP: .BLKW   2         ;NAME OF ACP FOR DEVICE
000054  B.MLEN=.                  ;LENGTH OF MOUNT TRANSACTION


        ;
        ; STATUS BITS FOR MOUNT STATUS MASK (B.DMST)
        ;
        BM.SHR=1                  ;DEVICE IS MOUNTED SHARED
        BM.NOS=2                  ;DEVICE IS MOUNTED NOSHARE
        BM.SYS=4                  ;DEVICE IS MOUNTED FOR THE SYSTEM (PUBLIC)
        BM.FOR=10                 ;DEVICE IS MOUNTED FOREIGN
```

ACNDF$ (Cont.)

```
        ;
        ; SYSTEM TIME CHANGE TRANSACTION
        ;
        .=$$$HLN                        ;
000022  B.TOLD: .BLKB   6               ;OLD TIME (YR, MON, DAY, HR, MIN, SEC)
000030  B.TNEW: .BLKB   6               ;NEW TIME (YR, MON, DAY, HR, MIN, SEC)
000036  B.TMLN=.                        ;TRANSACTION LENGTH


        ;
        ; PRINT DESPOOLER TRANSACTION
        ;
        .=$$$HLN                        ;START AFTER HEADER
000022  B.PNAM: .BLKW   3               ;PRINT JOB NAME  (RAD50)
000030  B.PPGS: .BLKW   1               ;PAGE COUNT
000032  B.PNFI: .BLKW   1               ;NUMBER OF FILES PRINTED
000034  B.PFRM: .BLKB   1               ;FORM NUMBER
000035  B.PPRI: .BLKB   1               ;PRINT PRIORITY
000036  B.PDEV: .BLKW   1               ;PRINT DEVICE NAME  (ASCII)
000040  B.PPUN: .BLKB   1               ;UNIT NUMBER OF PRINT DEVICE
000041  B.PLEN=.                        ;TRANSACTION LENGTH


        ;
        ; CARD READER SPOOLING TRANSACTION
        ;
        .=$$$HLN                        ;START AFTER HEADER
000022  B.RNAM: .BLKW   3               ;BATCH OR PRINT JOB NAME
000030  B.RCDS: .BLKW   1               ;NUMBER OF CARDS READ
000032  B.RDEV: .BLKW   1               ;READER DEVICE NAME (ASCII)
000034  B.RUNT: .BLKB   1               ;UNIT NUMBER OF READER DEVICE
000035  B.RSOP: .BLKB   1               ;SUBMIT OR PRINT (0=SUBMIT, 1=PRINT)
000036  B.RLEN=.                        ;TRANSACTION LENGTH


        ;
        ; LOGIN TRANSACTION
        ;
        .=$$$HLN                        ;START AFTER HEADER
000022  B.LUIC: .BLKW   1               ;LOGIN UIC
000024  B.LNAM: .BLKB   14.             ;USER'S LAST NAME
000042          .BLKB   1               ;AND FIRST INITIAL
000043  B.LLEN=.                        ;TRANSACTION LENGTH


        ;
        ;  RESET TRANSACTION PARAMETERS
        ;
        .=$$$HLN                        ;AFTER HEADER
000022  B.OFID: .BLKW   3               ;FILE-ID OF OLD TRN. FILE
000030  B.ODNM: .BLKB   2               ;DEVICE OF OLD TRN. FILE
000032  B.OUNT: .BLKW   1               ;UNIT OF OLD TRN. FILE
000034  B.NFID: .BLKW   3               ;FILE-ID OF NEW TRN. FILE
000042  B.NDNM: .BLKB   2               ;DEVICE OF NEW TRN. FILE
000044  B.NUNT: .BLKW   1               ;UNIT OF NEW TRN. FILE
000046  B.OEXS: .BLKW   1               ;EXT. SIZE FOR OLD TRN. FILE
000050  B.NEXS: .BLKW   1               ;EXT. SIZE FOR NEW TRN. FILE
000052  B.OSCR: .BLKW   1               ;OLD SCAN RATE IN SECONDS
000054  B.NSCR: .BLKW   1               ;NEW SCAN RATE IN SECONDS
```

ACNDF$ (Cont.)

```
000056  B.ODSC: .BLKW    1        ;OLD STATISTICAL SCAN RATE
000060  B.NDSC: .BLKW    1        ;NEW STATISTICAL SCAN RATE
000062  B.RTLN=.


        ;
        ; TRANSACTION TYPES
        ;
        ;       000 THRU 127            RESERVED FOR DEC USE
        ;       128 THRU 255            RESERVED FOR CUSTOMER USE
        ;
        BT.SAB=1                  ;SYSTEM ACCOUNT BLOCK (SAB)
        BT.UAB=2                  ;USER ACCOUNT BLOCK (UAB)
        BT.TAB=3                  ;TASK ACCOUNT BLOCK (TAB)
        BT.SS=11                  ;SYSLOG STARTUP TRANSACTION
        BT.INV=12                 ;INVALID LOGIN TRANSACTION
        BT.TIM=13                 ;SYSTEM TIME CHANGE TRANSACTION
        BT.ALL=14                 ;ALLOCATE DEVICE TRANSACTION
        BT.DEA=15                 ;DEALLOCATE DEVICE TRANSACTION
        BT.MOU=16                 ;MOUNT DEVICE TRANSACTION
        BT.DMO=17                 ;DISMOUNT DEVICE TRANSACTION
        BT.PRT=20                 ;PRINT DESPOOLER TRANSACTION
        BT.DIR=21                 ;DISK ACCOUNTING BY DIRECTORY  (UNSUPPORTED)
        BT.VOL=22                 ;DISK ACCOUNTING BY VOLUME     (UNSUPPORTED)
        BT.LOG=23                 ;LOGIN TRANSACTION
        BT.CRH=24                 ;CRASH RECOVERY TRANSACTION
        BT.DST=25                 ;DEVICE STATISTICS (UCB EXTENSION)
        BT.RTP=26                 ;RESET TRANSACTION PARAMETERS
        BT.INP=27                 ;CARD READER SPOOLING TRANSACTION


        ;
        ; STATUS MASK BIT DEFINITIONS (B.STM)
        ;
        BS.ACT=200                ;CONTROL BLOCK ACTIVE
        BS.CRH=100                ;RECORD FROM "TMP" FILE AFTER SYSTEM CRASH
        BS.LGO=40                 ;LOGGED OFF WITH OUTSTANDING ACTIVITY (UAB)
        BS.CO=40                  ;TASK'S TI: IS CO: (TAB ONLY)
        BS.TML=20                 ;TAB EXISTS ONLY FOR TIME LIMIT (TAB ONLY)
        BS.ZER=10                 ;LAST CPU INTERVAL WAS OF LENGTH ZERO
        BS.SCN=4                  ;TRANSACTION READY FOR WRITE TO SCAN FILE


        ;
        ; ACCOUNTING FEATURE MASK ($ACNFE)
        ;
        BF.DST=40000              ;STATISTICAL SCAN RATE
        BF.WRT=2000               ;FORCE SYSLOG TO WRITE ITS BUFFER
        BF.SCN=1000               ;SCAN REQUESTED
        BF.SLR=400                ;SYSLOG IS RUNNING (NOT STOPPED)
        BF.ERR=200                ;ACCOUNTING STOPPED DUE TO FATAL ERROR
        BF.STR=100                ;ACCOUNTING IS STARTING UP / SHUTTING DOWN
        BF.LSS=40                 ;ACCUMULATE SYSTEM STATISTICS
                                  ;(POINT UAB TO SAB)
        BF.TRN=10                 ;OUTPUT TO TRANSACTION FILE
        BF.XTK=4                  ;CHECKPOINT REQUEST IS DUE TO EXTK$
        BF.TSK=2                  ;TASK ACCOUNTING TURNED ON
        BF.XAC=1                  ;EXTENDED ACCOUNTING ASSEMBLED IN
```

ACNDF$ (Cont.)

```
        ;
        ; SHUTDOWN CODES (B.SHDN)
        ;
        ; 1             MAINTENANCE
        ; 2             REBOOT
        ; 3             SCHEDULED_SHUTDOWN
        ; 4             ACCOUNTING SHUTDOWN BY TASK "SHUTUP"
        ; 5             OTHER


        B.MAXL=128.             ;MAXIMUM TRANSACTION LENGTH
        B.MINL=$$$HLN           ;MINIMUM TRANSACTION LENGTH

                .PSECT
```

ACTDF$

```
                ACTDF$


                .ASECT
        .=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       ;DATE OF 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       ;RAD50 USER CLI
000070          .BLKW   2       ;UNUSED
000074  A.LPRV: .BLKW   1       ;LOGIN PRIVILEGE WORD
000076  A.SID:  .BLKW   1       ;SESSION IDENTIFIER
        A.LEN=128.              ;LENGTH OF CONTROL BLOCK


        ;
        ; BIT DEFINITIONS ON A.LPRV - LOGIN PRIVILEGE BITS
        ;
        AL.SLV=1                ;SLAVE TERMINAL ON LOGIN

                .PSECT
```

CLKDF$

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


        ;
        ; CLOCK QUEUE CONTROL BLOCK TYPE INDEPENDENT OFFSET 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 ADR OR SYSTEM SUBROUTINE IDENTIFICATION
000006  C.TIM:  .BLKW   2       ;ABSOLUTE TIME WHEN REQUEST COMES DUE


        ;
        ; CLOCK QUEUE CONTROL BLOCK-MARK TIME DEPENDENT OFFSET DEFINITIONS
        ;
        .=C.TIM+4               ;START OF DEPENDENT AREA
000012  C.AST:  .BLKW   1       ;AST ADDRESS
000014  C.SRC:  .BLKW   1       ;FLAG MASK WORD FOR 'BIS' SOURCE
000016  C.DST:  .BLKW   1       ;ADDRESS OF 'BIS' DESTINATION
000020          .BLKW   1       ;UNUSED


        ;
        ; CLOCK QUEUE CONTROL BLOCK-PERIODIC RESCHEDULING DEPENDENT OFFSET
        ; DEFINITIONS
        ;
        .=C.TIM+4               ;START OF DEPENDENT AREA
000012  C.RSI:  .BLKW   2       ;RESCHEDULE INTERVAL IN CLOCK TICKS
000016  C.UIC:  .BLKW   1       ;SCHEDULING UIC
000020  C.UAB:  .BLKW   1       ;POINTER TO ASSOCIATED UAB


        ;
        ; CLOCK QUEUE CONTROL BLOCK-SINGLE SHOT DEPENDENT OFFSET DEFINITIONS
        ;
        .=C.TIM+4               ;START OF DEPENDENT AREA
000012          .BLKW   2       ;TWO UNUSED WORDS
```

CLKDF$ (Cont.)

```
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.
        ;
        .=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

                .PSECT
```

CTBDF$

```
                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 $INTSV/$INTSE.
        ;
                .ASECT
        .=177756
177756  L.CLK:  .BLKW   8.      ;START OF CLOCK BLOCK (IF ANY)
177776  L.ICB:  .BLKW   1       ;ICB CHAIN FOR THIS CTB
000000  L.LNK:  .BLKW   1       ;CTB LINK WORD
000002  L.NAM:  .BLKW   1       ;GENERIC CONTROLLER NAME (ASCII)
000004  L.DCB:  .BLKW   1       ;DCB ADDRESS OF THIS DEVICE
000006  L.NUM:  .BLKB   1       ;NUMBER OF KRB ADDRESSES IN TABLE
000007  L.STS:  .BLKB   1       ;CTB STATUS BYTE
000010  L.KRB:  .BLKW   1       ;START OF KRB ADDRESSES


        ;
        ; NOTE: THE SYMBOL $XYCTB:: IS DEFINED FOR EACH CTB, WHERE THE
        ; CHARACTERS XY ARE THE SAME AS THOSE STORED IN L.NAM.  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).
        ;
                .PSECT


        ;
        ; CONTROLLER TABLE STATUS BYTE BIT DEFINITIONS
        ;
        LS.CLK=1                ;CLOCK BLOCK AT TOP OF CTB (1=YES)
        LS.MDC=2                ;MULTIDRIVER CTB (1=YES)
        LS.CBL=4                ;CLOCK BLK LINKED INTO CLK Q (1=YES)
        LS.CIN=10               ;CONT. USE COMMON INT TABLE (1=YES)
        LS.NET=20               ;THIS IS DECNET DEVICE.  ICB'S IN K.PRM
                                ;(1=YES)


        ;
        ; COMMON INTERRUPT TABLE DISPATCH ENTRY POINTS
        ;
        CI.CSR=-6               ;CSR TEST ENTRY POINT
        CI.KRB=-4               ;KRB STATUS CHANGE ENTRY POINT
        CI.PWF=-2               ;POWERFAIL ENTRY POINT
        CI.INT=0                ;COMMON INTERRUPT ADDRESS
        CI.DCB=2                ;START OF DCB TABLE (0 ENDS TABLE)
```

DCBDF$

```
                DCBDF$  ,,SYSDEF

        ;
        ; DEVICE CONTROL BLOCK
        ;
        ; THE DEVICE CONTROL BLOCK (DCB) DEFINES GENERIC INFORMATION ABOUT A
        ; DEVICE TYPE AND THE LOWEST AND HIGHEST UNIT NUMBERS.  THERE IS AT
        ; LEAST ONE DCB FOR EACH DEVICE TYPE IN A SYSTEM.  FOR EXAMPLE, IF
        ; THERE ARE TELETYPES IN A SYSTEM, THEN THERE IS AT LEAST ONE DCB
        ; WITH THE DEVICE NAME 'TT'.  IF PART OF THE TELETYPES WERE
        ; INTERFACED VIA DL11-A'S AND THE REST VIA A DH11, THEN THERE WOULD
        ; BE TWO DCB'S.  ONE FOR ALL DL11-A INTERFACED TELETYPES, AND ONE
        ; FOR ALL DH11 INTERFACED TELETYPES.
        ;
                .ASECT
        .=0
000000  D.LNK:  .BLKW   1       ;LINK TO NEXT DCB
000002  D.UCB:  .BLKW   1       ;POINTER TO FIRST UNIT CONTROL BLOCK
000004  D.NAM:  .BLKW   1       ;GENERIC DEVICE NAME
000006  D.UNIT: .BLKB   1       ;LOWEST UNIT NUMBER COVERED BY THIS DCB
000007          .BLKB   1       ;HIGHEST UNIT NUMBER COVERED BY THIS DCB
000010  D.UCBL: .BLKW   1       ;LENGTH OF EACH UNIT CONTROL BLOCK IN BYTES
000012  D.DSP:  .BLKW   1       ;POINTER TO DRIVER DISPATCH TABLE
000014  D.MSK:  .BLKW   1       ;LEGAL FUNCTION MASK  CODES 0-15.
000016          .BLKW   1       ;CONTROL FUNCTION MASK CODES 0-15.
000020          .BLKW   1       ;NOP'ED FUNCTION MASK CODES 0-15.
000022          .BLKW   1       ;ACP FUNCTION MASK CODES 0-15.
000024          .BLKW   1       ;LEGAL FUNCTION MASK CODES 16.-31.
000026          .BLKW   1       ;CONTROL FUNCTION MASK CODES 16.-31.
000030          .BLKW   1       ;NOP'ED FUNCTION MASK CODES 16.-31.
000032          .BLKW   1       ;ACP FUNCTION MASK CODES 16.-31.
000034  D.PCB:  .BLKW   1       ;LOADABLE DRIVER PCB ADDRESS

                .PSECT


        ;
        ; DRIVER DISPATCH TABLE OFFSET DEFINITIONS
        ;
        D.VDEB=-2               ;DEALLOCATE BUFFER(S)
        D.VCHK=-4               ;ADDRESS OF ROUTINE CALLED TO VALIDATE
                                ;AND CONVERT THE LBN.  USED BY DRIVERS
                                ;THAT SUPPORT SEEK OPTIMIZATION.
        D.VNXC=-4               ;ADDRESS OF ROUTINE IN TTDRV CALLED TO
                                ;HAVE IT SEND THE NEXT COMMAND IN THE
                                ;TYPEAHEAD BUFFER TO MCR...
        D.VINI=0                ;DEVICE INITIATOR
        D.VCAN=2                ;CANCEL CURRENT I/O FUNCTION
        D.VOUT=4                ;DEVICE TIMEOUT
        D.VPWF=6                ;POWERFAIL RECOVERY
        D.VKRB=10               ;CONTROLLER STATUS CHANGE ENTRY
        D.VUCB=12               ;UNIT STATUS CHANGE ENTRY


                .IF NB SYSDEF

        D.VINT=14               ;BEGINNING OF INTERRUPT STUFF

                .ENDC
```

EPKDF$

```
                 EPKDF$

         ;
         ; ERROR MESSAGE BLOCK DEFINITIONS
         ;
                 .ASECT


         ;
         ; HEADER SUBPACKET
         ;
         ;
         ;       +-----------------------------------------------+
         ;       | SUBPACKET LENGTH IN BYTES                     |
         ;       +-----------------------------------------------+
         ;       | SUBPACKET FLAGS                               |
         ;       +-----------------------+-----------------------+
         ;       | FORMAT IDENTIFICATION | OPERATING SYSTEM CODE |
         ;       +-----------------------+-----------------------+
         ;       | OPERATING SYSTEM IDENTIFICATION               |
         ;       |                                               |
         ;       +-----------------------+-----------------------+
         ;       | FLAGS                 | CONTEXT CODE          |
         ;       +-----------------------+-----------------------+
         ;       | ENTRY SEQUENCE                                |
         ;       +-----------------------------------------------+
         ;       | ERROR SEQUENCE                                |
         ;       +-----------------------+-----------------------+
         ;       | ENTRY TYPE SUBCODE    | ENTRY TYPE CODE       |
         ;       +-----------------------+-----------------------+
         ;       | TIME STAMP                                    |
         ;       |                                               |
         ;       |                                               |
         ;       +-----------------------+-----------------------+
         ;       | RESERVED              | PROCESSOR TYPE        |
         ;       +-----------------------+-----------------------+
         ;       | PROCESSOR IDENTIFICATION (URM)                |
         ;       +-----------------------------------------------+
         ;
         .=0
000000   E$HLGH: .BLKW   1       ; SUBPACKET LENGTH IN BYTES
000002   E$HSBF: .BLKW   1       ; SUBPACKET FLAGS
000004   E$HSYS: .BLKB   1       ; OPERATING SYSTEM CODE
000005   E$HIDN: .BLKB   1       ; FORMAT IDENTIFICATION
000006   E$HSID: .BLKB   4       ; OPERATING SYSTEM IDENTIFICATION
000012   E$HCTX: .BLKB   1       ; CONTEXT CODE
000013   E$HFLG: .BLKB   1       ; FLAGS
000014   E$HENS: .BLKW   1       ; ENTRY SEQUENCE NUMBER
000016   E$HERS: .BLKW   1       ; ERROR SEQUENCE NUMBER
000020   E$HENC:                 ; ENTRY CODE
000020   E$HTYC: .BLKB   1       ; ENTRY TYPE CODE
000021   E$HTYS: .BLKB   1       ; ENTRY TYPE SUBCODE
000022   E$HTIM: .BLKB   6       ; TIME STAMP
000030   E$HPTY: .BLKB   1       ; PROCESSOR TYPE
000031           .BLKB   1       ; RESERVED
000032   E$HURM: .BLKW   1       ; PROCESSOR IDENTIFICATION (URM)
                 .EVEN
000034   E$HLEN:                 ; LENGTH
```

EPKDF$ (Cont.)

```
;
; SUBPACKET FLAGS FOR E$HSBF
;
        SM.ERR  =      1 ; ERROR PACKET
        SM.HDR  =      1 ; HEADER SUBPACKET
        SM.TSK  =      2 ; TASK SUBPACKET
        SM.DID  =      4 ; DEVICE IDENTIFICATION SUBPACKET
        SM.DOP  =     10 ; DEVICE OPERATION SUBPACKET
        SM.DAC  =     20 ; DEVICE ACTIVITY SUBPACKET
        SM.DAT  =     40 ; DATA SUBPACKET
        SM.MBC  =  20000 ; 22-BIT MASSBUS CONTROLLER PRESENT
        SM.CMD  =  40000 ; ERROR LOG COMMAND PACKET
        SM.ZER  =100000 ; ZERO I/O COUNTS

;
; CODES FOR FIELD E$HIDN
;
        EH$FOR  =      1 ; CURRENT PACKET FORMAT

;
; FLAGS FOR THE ERROR LOG FLAGS BYTE ($ERFLA) IN THE EXEC
;
        ES.INI  =      1 ; ERROR LOG INITIALIZED
        ES.DAT  =      2 ; ERROR LOG RECEIVING DATA PACKETS
        ES.LIM  =      4 ; ERROR LIMITING ENABLED
        ES.LOG  =     10 ; ERROR LOGGING ENABLED

;
; TYPE AND SUBTYPE CODES FOR FIELDS E$HTYC AND E$HTYS
;
;       SYMBOLS WITH NAMES E$CXXX ARE TYPE CODES FOR FIELD E$HTYC,
;       SYMBOLS WITH NAMES E$SXXX ARE SUBTYPE CODES FOR FIELD E$HTYS
;
        E$CCMD  =      1 ; ERROR LOG CONTROL
        E$SSTA  =      1 ;       ERROR LOG STATUS CHANGE
        E$SSWI  =      2 ;       SWITCH LOGGING FILES
        E$SAPP  =      3 ;       APPEND FILE
        E$SBAC  =      4 ;       DECLARE BACKUP FILE
        E$SSHO  =      5 ;       SHOW
        E$SCHL  =      6 ;       CHANGE LIMITS

        E$CERR  =      2 ; DEVICE ERRORS
        E$SDVH  =      1 ;       DEVICE HARD ERROR
        E$SDVS  =      2 ;       DEVICE SOFT ERROR
        E$STMO  =      3 ;       DEVICE INTERRUPT TIMEOUT
        E$SUNS  =      4 ;       DEVICE UNSOLICITED INTERRUPT

        E$CDVI  =      3 ; DEVICE INFORMATION
        E$SDVI  =      1 ;       DEVICE INFORMATION MESSAGE

        E$CDCI  =      4 ; DEVICE CONTROL INFORMATION
        E$SMOU  =      1 ;       DEVICE MOUNT
        E$SDMO  =      2 ;       DEVICE DISMOUNT
        E$SRES  =      3 ;       DEVICE COUNT RESET
        E$SRCT  =      4 ;       BLOCK REPLACEMENT

        E$CCPU  =      5 ; CPU DETECTED ERRORS
        E$SMEM  =      1 ;       MEMORY ERROR
        E$SINT  =      2 ;       UNEXPECTED INTERRUPT
```

EPKDF$ (Cont.)

```
              E$CSYS  =     6 ; SYSTEM CONTROL INFORMATION
              E$SPWR  =     1 ;      POWER RECOVERY

              E$CCTL  =     7 ; CONTROL INFORMATION
              E$STIM  =     1 ;      TIME CHANGE
              E$SCRS  =     2 ;      SYSTEM CRASH
              E$SLOA  =     3 ;      DEVICE DRIVER LOAD
              E$SUNL  =     4 ;      DEVICE DRIVER UNLOAD
              E$SHRC  =     5 ;      RECONFIGURATION STATUS CHANGE
              E$SMES  =     6 ;      MESSAGE

              E$CSDE  =    10 ; SOFTWARE DETECTED EVENTS
              E$SABO  =     1 ;      TASK ABORT

       ;
       ; CODES FOR CONTEXT CODE ENTRY E$HCTX
       ;
              EH$NOR  =     1 ; NORMAL ENTRY
              EH$STA  =     2 ; START ENTRY
              EH$CRS  =     3 ; CRASH ENTRY

       ;
       ; CODES FOR FLAGS ENTRY E$HFLG
       ;
              EH$VIR  =     1 ; ADDRESSES ARE VIRTUAL
              EH$EXT  =     2 ; ADDRESSES ARE EXTENDED
              EH$COU  =     4 ; ERROR COUNTS SUPPLIED


       ;
       ; TASK SUBPACKET
       ;
       ;      +-----------------------------------------------+
       ;      | TASK SUBPACKET LENGTH                         |
       ;      +-----------------------------------------------+
       ;      | TASK NAME IN RAD50                            |
       ;      |                                               |
       ;      +-----------------------------------------------+
       ;      | TASK UIC                                      |
       ;      +-----------------------------------------------+
       ;      | TASK TI: DEVICE NAME                          |
       ;      +-----------------------+-----------------------+
       ;      | FLAGS                 | TASK TI: UNIT NUMBER  |
       ;      +-----------------------+-----------------------+
       ;
       .=0
000000 E$TLGH: .BLKW   1        ; TASK SUBPACKET LENGTH
000002 E$TTSK: .BLKW   2        ; TASK NAME IN RAD50
000006 E$TUIC: .BLKW   1        ; TASK UIC
000010 E$TTID: .BLKB   2        ; TASK TI: DEVICE NAME
000012 E$TTIU: .BLKB   1        ; TASK TI: UNIT
000013 E$TFLG: .BLKB   1        ; FLAGS
               .EVEN
000014 E$TLEN:

       ;
       ; FLAGS FOR ENTRY E$TFLG
       ;
              ET$PRV  =     1 ; TASK IS PRIVILEGED
              ET$PRI  =     2 ; TERMINAL IS PRIVILEGED
```

EPKDF$ (Cont.)

```
        ;
        ; DEVICE IDENTIFICATION SUBPACKET
        ;
        ;       +-----------------------------------------------+
        ;       | DEVICE IDENTIFICATION SUBPACKET LENGTH        |
        ;       +-----------------------------------------------+
        ;       | DEVICE MNEMONIC NAME                          |
        ;       +-----------------------+-----------------------+
        ;       | CONTROLLER NUMBER     | DEVICE UNIT NUMBER    |
        ;       +-----------------------+-----------------------+
        ;       | PHYSICAL SUBUNIT #    | PHYSICAL UNIT #       |
        ;       +-----------------------+-----------------------+
        ;       | PHYSICAL DEVICE MNEMONIC (RSX-11M-PLUS ONLY)  |
        ;       +-----------------------+-----------------------+
        ;       | RESERVED              | FLAGS                 |
        ;       +-----------------------+-----------------------+
        ;       | VOLUME NAME OF MOUNTED VOLUME                 |
        ;       |                                               |
        ;       |                                               |
        ;       |                                               |
        ;       |                                               |
        ;       |                                               |
        ;       +-----------------------------------------------+
        ;       | PACK IDENTIFICATION                           |
        ;       |                                               |
        ;       +-----------------------------------------------+
        ;       | DEVICE TYPE CLASS                             |
        ;       +-----------------------------------------------+
        ;       | DEVICE TYPE                                   |
        ;       |                                               |
        ;       +-----------------------------------------------+
        ;       | I/O OPERATION COUNT LONGWORD                  |
        ;       |                                               |
        ;       +-----------------------+-----------------------+
        ;       | HARD ERROR COUNT      | SOFT ERROR COUNT      |
        ;       +-----------------------+-----------------------+
        ;       | BLOCKS TRANSFERRED COUNT (RSX-11M-PLUS ONLY)  |
        ;       |                                               |
        ;       +-----------------------------------------------+
        ;       | CYLINDERS CROSSED COUNT (RSX-11M-PLUS ONLY)   |
        ;       |                                               |
        ;       +-----------------------------------------------+
        ;
        .=0
000000  E$ILGH: .BLKW   1       ; DEVICE IDENTIFICATION SUBPACKET LENGTH
000002  E$ILDV: .BLKW   1       ; DEVICE MNEMONIC NAME
000004  E$ILUN: .BLKB   1       ; DEVICE UNIT NUMBER
000005  E$IPCO: .BLKB   1       ; CONTROLLER NUMBER
000006  E$IPUN: .BLKB   1       ; PHYSICAL UNIT NUMBER
000007  E$IPSU: .BLKB   1       ; PHYSICAL SUBUNIT NUMBER
000010  E$IPDV: .BLKW   1       ; PHYSICAL DEVICE MNEMONIC
000012  E$IFLG: .BLKB   1       ; FLAGS
000013          .BLKB   1       ; RESERVED
000014  E$IVOL: .BLKB   12.     ; VOLUME NAME
000030  E$IPAK: .BLKB   4       ; PACK IDENTIFICATION
000034  E$IDEV:                 ; DEVICE TYPE
000034  E$IDCL: .BLKW   1       ; DEVICE TYPE CLASS
000036  E$IDTY: .BLKW   2       ; DEVICE TYPE
000042  E$IOPR: .BLKW   2       ; I/O OPERATION COUNT LONGWORD
000046  E$IERS: .BLKB   1       ; SOFT ERROR COUNT
000047  E$IERH: .BLKB   1       ; HARD ERROR COUNT
```

EPKDF$ (Cont.)

```
000050  E$IBLK: .BLKW   2       ; BLOCKS TRANSFERRED COUNT
000054  E$ICYL: .BLKW   2       ; CYLINDERS CROSSED COUNT
                .EVEN
000060  E$ILEN:                 ; SUBPACKET LENGTH

        ;
        ; FLAGS FOR FIELD E$IFLG
        ;
                EI$SUB  =      1 ; SUBCONTROLLER DEVICE
                EI$NUX  =      2 ; NO UCB EXTENSION, DATA INVALID


        ;
        ; DEVICE OPERATION SUBPACKET
        ;
        ;       +------------------------------------------------+
        ;       | DEVICE OPERATION SUBPACKET LENGTH              |
        ;       +------------------------------------------------+
        ;       | TASK NAME IN RAD50                             |
        ;       |                                                |
        ;       +------------------------------------------------+
        ;       | TASK UIC                                       |
        ;       +------------------------------------------------+
        ;       | TASK TI: LOGICAL DEVICE MNEMONIC               |
        ;       +-----------------------+------------------------+
        ;       | RESERVED              | TASK TI: DEVICE UNIT   |
        ;       +-----------------------+------------------------+
        ;       | I/O FUNCTION CODE                              |
        ;       +-----------------------+------------------------+
        ;       | RESERVED              | OPERATION FLAGS        |
        ;       +-----------------------+------------------------+
        ;       | TRANSFER OPERATION ADDRESS                     |
        ;       |                                                |
        ;       +------------------------------------------------+
        ;       | TRANSFER OPERATION BYTE COUNT                  |
        ;       +------------------------------------------------+
        ;       | CURRENT OPERATION RETRY COUNT                  |
        ;       +------------------------------------------------+
        ;
        .=0
000000  E$OLGN: .BLKW   1       ; SUBPACKET LENGTH
000002  E$OTSK: .BLKW   2       ; TASK NAME IN RAD50
000006  E$OUIC: .BLKW   1       ; TASK UIC
000010  E$OTID: .BLKB   2       ; TASK TI: LOGICAL DEVICE MNEMONIC
000012  E$OTIU: .BLKB   1       ; TASK TI: LOGICAL DEVICE UNIT
000013          .BLKB   1       ; RESERVED
000014  E$OFNC: .BLKW   1       ; I/O FUNCTION CODE
000016  E$OFLG: .BLKB   1       ; OPERATION FLAGS
000017          .BLKB   1       ; RESERVED
000020  E$OADD: .BLKW   2       ; TRANSFER OPERATION ADDRESS
000024  E$OSIZ: .BLKW   1       ; TRANSFER OPERATION BYTE COUNT
000026  E$ORTY: .BLKW   1       ; CURRENT OPERATION RETRY COUNT
                .EVEN
000030  E$OLEN:                 ; DEVICE OPERATION SUBPACKET LENGTH

        ;
        ; FLAGS FOR FIELD E$OFLG
        ;
                EO$TRA  =      1 ; TRANSFER OPERATION
                EO$DMA  =      2 ; DMA DEVICE
```

EPKDF$ (Cont.)

```
                EO$EXT  =     4 ; EXTENDED ADDRESSING DEVICE
                EO$PIP  =    10 ; DEVICE IS POSITIONING


        ;
        ; I/O ACTIVITY SUBPACKET
        ;
        ;       +-----------------------------------------------+
        ;       | I/O ACTIVITY SUBPACKET LENGTH                 |
        ;       +-----------------------------------------------+
        ;
        .=0
000000  E$ALGH: .BLKW   1       ; SUBPACKET LENGTH


        ;
        ; I/O ACTIVITY SUBPACKET ENTRY
        ;
        ;       +-----------------------------------------------+
        ;       | LOGICAL DEVICE NAME MNEMONIC                  |
        ;       +-----------------------+-----------------------+
        ;       | CONTROLLER NUMBER     | LOGICAL DEVICE UNIT   |
        ;       +-----------------------+-----------------------+
        ;       | PHYSICAL SUBUNIT #    | PHYSICAL UNIT NUMBER  |
        ;       +-----------------------+-----------------------+
        ;       | PHYSICAL DEVICE MNEMONIC (RSX-11M-PLUS ONLY)  |
        ;       +-----------------------+-----------------------+
        ;       | TASK TI: LOGICAL UNIT | DEVICE FLAGS          |
        ;       +-----------------------+-----------------------+
        ;       | REQUESTING TASK NAME IN RAD50                 |
        ;       |                                               |
        ;       +-----------------------------------------------+
        ;       | REQUESTING TASK UIC                           |
        ;       +-----------------------------------------------+
        ;       | TASK TI: LOGICAL DEVICE NAME                  |
        ;       +-----------------------------------------------+
        ;       | I/O FUNCTION CODE                             |
        ;       +-----------------------+-----------------------+
        ;       | RESERVED              | FLAGS                 |
        ;       +-----------------------+-----------------------+
        ;       | TRANSFER OPERATION ADDRESS                    |
        ;       |                                               |
        ;       +-----------------------------------------------+
        ;       | TRANSFER OPERATION BYTE COUNT                 |
        ;       +-----------------------------------------------+
        ;
        .=0
000000  E$ALDV: .BLKW   1       ; LOGICAL DEVICE NAME MNEMONIC
000002  E$ALUN: .BLKB   1       ; LOGICAL DEVICE UNIT
000003  E$APCO: .BLKB   1       ; CONTROLLER NUMBER
000004  E$APUN: .BLKB   1       ; PHYSICAL UNIT NUMBER
000005  E$APSU: .BLKB   1       ; PHYSICAL SUBUNIT NUMBER
000006  E$APDV: .BLKW   1       ; PHYSICAL DEVICE MNEMONIC
000010  E$ADFG: .BLKB   1       ; DEVICE FLAGS
000011  E$ATIU: .BLKB   1       ; TASK TI: LOGICAL UNIT
000012  E$ATSK: .BLKW   2       ; REQUESTING TASK NAME IN RAD50
000016  E$AUIC: .BLKW   1       ; REQUESTING TASK UIC
000020  E$ATID: .BLKW   1       ; TASK TI: LOGICAL DEVICE NAME
000022  E$AFNC: .BLKW   1       ; I/O FUNCTION CODE
000024  E$AFLG: .BLKB   1       ; FLAGS
```

EPKDF$ (Cont.)

```
000025          .BLKB   1         ; RESERVED
000026  E$AADD: .BLKW   2         ; TRANSFER OPERATION ADDRESS
000032  E$ASIZ: .BLKW   1         ; TRANSFER OPERATION BYTE COUNT
                .EVEN
000034  E$ALEN:                   ; SUBPACKET ENTRY LENGTH

        ;
        ; FLAGS FOR FIELD E$ADFG
        ;
                EA$SUB  =      1 ; SUBCONTROLLER DEVICE
                EA$NUX  =      2 ; NO UCB EXTENSION, DATA INVALID

        ;
        ; FLAGS FOR FIELD E$AFLG
        ;
                EA$TRA  =      1 ; TRANSFER OPERATION
                EA$DMA  =      2 ; DMA DEVICE
                EA$EXT  =      4 ; DEVICE HAS EXTENDED ADDRESSING
                EA$PIP  =     10 ; DEVICE IS POSITIONING

                .PSECT
```

F11DF$

```
                F11DF$  ,,SYSDEF

        ;
        ; VOLUME CONTROL BLOCK
        ;
                .ASECT
        .=0
000000  V.TRCT: .BLKW   1       ; TRANSACTION COUNT
000002  V.TYPE: .BLKB   1       ; VOLUME TYPE DESCRIPTOR
        VT.SL1= 1               ; FILES-11 STRUCTURE LEVEL 1
        VT.ANS= 10              ; ANSI LABELED TAPE
        VT.UNL= 11              ; UNLABELED TAPE
000003  V.VCHA: .BLKB   1       ; VOLUME CHARACTERISTICS
        VC.SLK= 1               ; CLEAR VOLUME VALID ON DISMOUNT
        VC.HLK= 2               ; UNLOAD THE VOLUME ON DISMOUNT
        VC.DEA= 4               ; DEALLOCATE THE VOLUME ON DISMOUNT
        VC.PUB= 10              ; SET (CLEAR) US.PUB ON DISMOUNT
000004  V.LABL: .BLKB   14      ; VOLUME LABEL (ASCII)
000020  V.PKSR: .BLKW   2       ; PACK SERIAL NUMBER FOR ERROR LOGGING

000024  V.SLEN:                 ; LENGTH OF SHORT VCB

000024  V.IFWI: .BLKW   1       ; INDEX FILE WINDOW
000026  V.FCB:  .BLKW   2       ; FILE CONTROL BLOCK LIST HEAD
000032  V.IBLB: .BLKB   1       ; INDEX BIT MAP 1ST LBN HIGH BYTE
000033  V.IBSZ: .BLKB   1       ; INDEX BIT MAP SIZE IN BLOCKS
000034          .BLKW   1       ; INDEX BITMAP 1ST LBN LOW BITS
000036  V.FMAX: .BLKW   1       ; MAX NO. OF FILES ON VOLUME
000040  V.WISZ: .BLKB   1       ; DEFAULT SIZE OF WINDOW IN RTRV PTRS
                                ; VALUE IS < 128.
000041  V.SBCL: .BLKB   1       ; STORAGE BIT MAP CLUSTER FACTOR
000042  V.SBSZ: .BLKW   1       ; STORAGE BIT MAP SIZE IN BLOCKS
000044  V.SBLB: .BLKB   1       ; STORAGE BIT MAP 1ST LBN HIGH BYTE
000045  V.FIEX: .BLKB   1       ; DEFAULT FILE EXTEND SIZE
000046          .BLKW   1       ; STORAGE BIT MAP 1ST LBN LOW BITS
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       ; NUMBER OF FREE BLOCKS ON VOLUME HIGH BYTE
000057  V.LRUC: .BLKB   1       ; COUNT OF AVAILABLE LRU SLOTS IN FCB LIST
000060          .BLKW   1       ; NUMBER OF FREE BLOCKS ON VOLUME LOW BITS
000062  V.STS:  .BLKB   1       ; VOL STATUS BYTE, CONTAINING THE FOLLOWING
        VS.IFW= 1               ; INDEX FILE IS WRITE ACCESSED
        VS.BMW= 2               ; STORAGE BITMAP FILE IS WRITE ACCESSED
000063  V.FFNU: .BLKB   1       ; FIRST FREE INDEX FILE BITMAP BLOCK
000064  V.EXT:  .BLKW   1       ; POINTER TO VCB EXTENSION

000066  V.LGTH:                 ; SIZE IN BYTES OF VCB


        ;
        ; MOUNT LIST ENTRY
        ;
        ; EACH ENTRY ALLOWS ACCESS TO A SPECIFIED USER FOR A NON-PUB DEVICE
        ;
        ; TO ALLOW EXPANSION, ONLY THE ONLY TYPE CODE DEFINED IS "1" FOR
        ; DEVICE ACCESS BLOCKS
        ;
                .ASECT
        .=0
000000  M.LNK:  .BLKW   1       ; LINK WORD
```

F11DF$ (Cont.)

```
000002  M.TYPE: .BLKB   1        ; TYPE OF ENTRY
        MT.MLS= 1                ; MOUNTED VOLUME USER ACCESS LIST
000003  M.ACC:  .BLKB   1        ; NUMBER OF ACCESSES
000004  M.DEV:  .BLKW   1        ; DEVICE UCB
000006  M.TI:   .BLKW   1        ; ACCESSOR TI: UCB

000010  M.LEN:                   ; LENGTH OF ENTRY


        ;
        ; FILE CONTROL BLOCK
        ;
                .ASECT
        .=0
000000  F.LINK: .BLKW   1        ; FCB CHAIN POINTER
000002  F.FNUM: .BLKW   1        ; FILE NUMBER
000004  F.FSEQ: .BLKW   1        ; FILE SEQUENCE NUMBER
000006          .BLKB   1        ; NOT USED
000007  F.FSQN: .BLKB   1        ; FILE SEGMENT NUMBER
000010  F.FOWN: .BLKW   1        ; FILE OWNER'S UIC
000012  F.FPRO: .BLKW   1        ; FILE PROTECTION CODE
000014  F.UCHA: .BLKB   1        ; USER CONTROLLED CHARACTERISTICS
000015  F.SCHA: .BLKB   1        ; SYSTEM CONTROLLED CHARACTERISTICS
000016  F.HDLB: .BLKW   2        ; FILE HEADER LOGICAL BLOCK NUMBER

                                 ; BEGINNING OF STATISTICS BLOCK
000022  F.LBN:  .BLKW   2        ; LBN OF VIRTUAL BLOCK 1 IF CONTIGUOUS
                                 ; 0 IF NON CONTIGUOUS
000026  F.SIZE: .BLKW   2        ; SIZE OF FILE IN BLOCKS
000032  F.NACS: .BLKB   1        ; NO. OF ACCESSES
000033  F.NLCK: .BLKB   1        ; NO. OF LOCKS

000012  S.STBK=.-F.LBN           ; SIZE OF STATISTICS BLOCK

000034  F.STAT:                  ; FCB STATUS WORD
000034  F.NWAC: .BLKB   1        ; NUMBER OF WRITE ACCESSORS
000035          .BLKB   1        ; STATUS BITS FOR FCB CONSISTING OF
        FC.WAC= 100000           ; SET IF FILE ACCESSED FOR WRITE
        FC.DIR= 40000            ; SET IF FCB IS IN DIRECTORY LRU
        FC.CEF= 20000            ; SET IF DIRECTORY EOF NEEDS UPDATING
        FC.FCO= 10000            ; SET IF TRYING TO FORCE DIRECTORY CONTIG
000036  F.DREF: .BLKW   1        ; DIRECTORY EOF BLOCK NUMBER
000040  F.DRNM: .BLKW   1        ; 1ST WORD OF DIRECTORY NAME
000042  F.FEXT: .BLKW   1        ; POINTER TO EXTENSION FCB
000044  F.FVBN: .BLKW   2        ; STARTING VBN OF THIS FILE SEGMENT
000050  F.LKL:  .BLKW   1        ; POINTER TO LOCKED BLOCK LIST FOR FILE
000052  F.WIN:  .BLKW   1        ; WINDOW BLOCK LIST FOR THIS FILE

000054  F.LGTH:                  ; SIZE IN BYTES OF FCB


        ;
        ; WINDOW
        ;
                .ASECT
        .=0
000000  W.ACT:                   ; NUMBER OF ACTIVE MAPPING POINTERS
                                 ;  WHEN NO SECONDARY POOL
```

F11DF$ (Cont.)

```
000000  W.BLKS:                 ; BLOCK SIZE OF SECONDARY POOL SEGMENT
                                ;  WHEN SECONDARY POOL
000000  W.CTL:  .BLKW   1       ; LOW BYTE = # OF MAP ENTRIES ACTIVE
                                ; HIGH BYTE CONSISTS OF CONTROL BITS
        WI.RDV= 400             ; READ VIRTUAL BLOCK ALLOWED IF SET
        WI.WRV= 1000            ; WRITE VIRTUAL BLOCK ALLOWED IF SET
        WI.EXT= 2000            ; EXTEND ALLOWED IF SET
        WI.LCK= 4000            ; SET IF LOCKED AGAINST SHARED ACCESS
        WI.DLK= 10000           ; SET IF DEACCESS LOCK ENABLED
        WI.PND= 20000           ; WINDOW TURN PENDING BIT
        WI.EXL= 40000           ; SET IF MANUAL UNLOCK DESIRED
        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.LKL:  .BLKW   1       ; POINTER TO LIST OF USERS LOCKED BLOCKS
000010  W.WIN:  .BLKW   1       ; WINDOW BLOCK LIST LINK WORD

                .IF NB SYSDEF   ; IF SYSDEF SPECIFIED IN CALL

                .IF NDF P$$WND  ; IF SECONDARY POOL WINDOWS NOT ALLOWED


        ;
        ; NON-SECONDARY POOL WINDOW BLOCK
        ;       IF SECONDARY POOL WINDOWS ARE NOT ENABLED, THE WINDOW BLOCK
        ;       CONTAINS THE CONTROL INFORMATION AND RETRIEVAL POINTERS.
        ;
        W.VBN:  .BLKB   1       ; HIGH BYTE OF 1ST VBN MAPPED BY WINDOW
        W.MAP:                  ; DEF LABEL WITH ODD ADDR 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

                .IFF            ; IF WINDOWS IN SECONDARY POOL


        ;
        ; SECONDARY POOL WINDOW CONTROL AND MAPPING BLOCK
        ;       IF SECONDARY POOL WINDOW BLOCKS ARE ENABLED, LUTN2 POINTS
        ;       TO A CONTROL BLOCK IN SYSTEM POOL WHICH CONTAINS THE
        ;       FOLLOWING CONTROL FIELDS AND THE MAPPING INFORMATION
        ;       FOR THE SECONDARY POOL WINDOW.
        ;
        W.MAP:  .BLKW   1       ; ADDR TO THE MAPPING PTRS IN SECONDARY POOL


        ;
        ; SECONDARY POOL WINDOW
        ;       IF SECONDARY POOL WINDOW BLOCKS ARE ENABLED, THE RETRIEVAL
        ;       POINTERS ARE MAINTAINED IN SECONDARY POOL IN THE FOLLOWING
        ;       FORMAT.
        ;
        .=0
                ASSUME  W.CTL,0
                .BLKB   1       ; NUMBER OF ACTIVE MAPPING POINTERS
        W.USE:  .BLKB   1       ; STATUS OF BLOCK
        W.VBN:  .BLKB   1       ; HIGH BYTE OF 1ST VBN MAPPED BY WINDOW
```

F11DF$ (Cont.)

```
        W.WISZ: .BLKB    1         ; SIZE IN RTRV PTRS OF WINDOW (7 BITS)
                .BLKW    1         ; LOW ORDER WORD OF 1ST VBN MAPPED
        W.RTRV:                    ; OFFSET TO 1ST RETRIEVAL POINTER IN WINDOW

                .ENDC ;P$$WND      ; END SECONDARY POOL WINDOW CONDITIONAL

                .ENDC ;SYSDEF      ; END SYSDEF CONDITIONAL


        ;
        ; LOCKED BLOCK LIST NODE
        ;
                .ASECT
        .=0
000000  L.LNK:  .BLKW    1         ; LINK TO NEXT NODE IN LIST
000002  L.WI1:  .BLKW    1         ; POINTER TO WINDOW FOR FIRST ENTRY
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:

                .PSECT
```

HDRDF$

```
                HDRDF$

        ;
        ; TASK HEADER OFFSET DEFINITIONS
        ;
                .ASECT
        .=0
000000  H.CSP:  .BLKW   1       ;CURRENT STACK POINTER
000002  H.HDLN: .BLKW   1       ;HEADER LENGTH IN BYTES
000004  H.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 ADDRESS
000036  H.RCVA: .BLKW   1       ;RECIEVE AST CONTROL BLOCK ADDRESS
000040  H.EFSV: .BLKW   1       ;EVENT FLAG ADDRESS SAVE ADDRESS
000042  H.FPSA: .BLKW   1       ;POINTER TO FLOATING POINT/EAE SAVE AREA
000044  H.WND:  .BLKW   1       ;POINTER TO NUMBER OF WINDOW BLOCKS
000046  H.DSW:  .BLKW   1       ;TASK DIRECTIVE STATUS WORD
000050  H.FCS:  .BLKW   1       ;FCS IMPURE POINTER
000052  H.FORT: .BLKW   1       ;FORTRAN IMPURE POINTER
000054  H.OVLY: .BLKW   1       ;OVERLAY IMPURE POINTER
000056  H.VEXT: .BLKW   1       ;WORK AREA EXTENSION VECTOR POINTER
000060  H.SPRI: .BLKB   1       ;PRIORITY DIFFERENCE FOR SWAPPING
000061  H.NML:  .BLKB   1       ;NETWORK MAILBOX LUN
000062  H.RRVA: .BLKW   1       ;RECEIVE BY REFERENCE AST CONTROL BLOCK 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
        ;
        .=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
```

HDRDF$ (Cont.)

```
000010  W.BSIZ: .BLKW   1        ;SIZE OF WINDOW IN 32W BLOCKS
000012  W.BOFF: .BLKW   1        ;PHYSICAL MEMORY OFFSET IN 32W BLOCKS
000014  W.BFPD: .BLKB   1        ;FIRST PDR ADDRESS
000015  W.BNPD: .BLKB   1        ;NUMBER OF PDR'S TO MAP
000016  W.BLPD: .BLKW   1        ;CONTENTS OF LAST PDR
000020  W.BLGH:                  ;LENGTH OF WINDOW DESCRIPTOR


        ;
        ; 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
```

HWDDF$

```
        HWDDF$  ,,SYSDEF

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


;
; EXTENDED ARITHMETIC ELEMENT REGISTERS
;
        .IF DF  E$$EAE

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

        .ENDC


;
; MEMORY MANAGEMENT HARDWARE REGISTERS AND STATUS CODES
;
        .IF NB  B

        CRESET  KINAR,172340    ;KERNEL I PAR'S
        CRESET  KINDR,172300    ;KERNEL I PDR'S
        CRESET  KDSAR,172360    ;KERNEL D PAR'S
        CRESET  KDSDR,172320    ;KERNEL D PDR'S
        CRESET  SISAR,172240    ;SUPERVISOR I PAR'S
        CRESET  SISDR,172200    ;SUPERVISOR I PDR'S
        CRESET  SDSAR,172260    ;SUPERVISOR D PAR'S
        CRESET  SDSDR,172220    ;SUPERVISOR D PDR'S
        CRESET  UINAR,177640    ;USER I PAR'S
```

HWDDF$ (Cont.)

```
        CRESET  UINDR,177600     ;USER I PDR'S
        CRESET  UDSAR,177660     ;USER D PAR'S
        CRESET  UDSDR,177620     ;USER D PDR'S

        .ENDC

        .IF NB  SYSDEF

        .IF DF  K$$DAS

        CRESET  KISAR,172360     ;KERNEL D PAR'S
        CRESET  KISDR,172320     ;KERNEL D PDR'S

        .IFF

        CRESET  KISAR,172340     ;KERNEL I PAR'S
        CRESET  KISDR,172300     ;KERNEL I PDR'S

        .ENDC

        .IF DF  U$$DAS

        CRESET  UISAR,177660     ;USER D PAR'S
        CRESET  UISDR,177620     ;USER D PDR'S

        .IFF    ; DF U$$DAS

        CRESET  UISAR,177640     ;USER I PAR'S
        CRESET  UISDR,177600     ;USER I PDR'S

        .ENDC   ; DF U$$DAS

        .ENDC

UBMPR=170200            ;UNIBUS MAPPING REGISTER 0
CMODE=140000            ;CURRENT MODE FIELD OF PS WORD
PMODE=30000             ;PREVIOUS MODE FIELD OF PS WORD
CSMODE=40000            ;CURRENT MODE = SUPERVISOR PS WORD BITS
PSMODE=10000            ;PREVIOUS MODE = SUPERVISOR PS WORD BITS
SR0=177572              ;SEGMENT STATUS REGISTER 0
SR3=172516              ;SEGMENT STATUS REGISTER 3
CPUERR=177766           ;CPU ERROR REGISTER
MEMERR=177744           ;MEMORY SYSTEM ERROR REGISTER
MEMCTL=177746           ;MEMORY CONTROL REGISTER


;
; FEATURE SYMBOL DEFINITIONS
;
FE.EXT=1                ;22-BIT EXTENDED MEMORY SUPPORT
FE.MUP=2                ;MULTI-USER PROTECTION SUPPORT
FE.EXV=4                ;EXECUTIVE IS SUPPORTED TO 20K
FE.DRV=10               ;LOADABLE DRIVER SUPPORT
FE.PLA=20               ;PLAS SUPPORT
FE.CAL=40               ;DYNAMIC CHECKPOINT SPACE ALLOCATION
FE.PKT=100              ;PREALLOCATION OF I/O PACKETS
FE.EXP=200              ;EXTEND TASK DIRECTIVE SUPPORTED
FE.LSI=400              ;PROCESSOR IS AN LSI-11
FE.OFF=1000             ;PARENT/OFFSPRING TASKING SUPPORTED
FE.FDT=2000             ;FULL DUPLEX TERMINAL DRIVER SUPPORTED
```

HWDDF$ (Cont.)

```
FE.X25=4000             ;X.25 CEX IS LOADED
FE.DYM=10000            ;DYNAMIC MEMORY ALLOCATION SUPPORTED
FE.CEX=20000            ;COM EXEC IS LOADED
FE.MXT=40000            ;MCR EXIT AFTER EACH COMMAND MODE
FE.NLG=100000           ;LOGINS DISABLED - MULTI-USER SUPPORT


;
; FEATURE MASK DEFINITIONS (SECOND WORD)
;
F2.DAS=1                ;KERNEL DATA SPACE SUPPORTED
F2.LIB=2                ;SUPERVISOR MODE LIBRARIES SUPPORTED
F2.MP=4                 ;SYSTEM SUPPORTS MULTIPROCESSING
F2.EVT=10               ;SYSTEM SUPPORTS EVENT TRACE FEATURE
F2.ACN=20               ;SYSTEM SUPPORTS CPU ACCOUNTING
F2.SDW=40               ;SYSTEM SUPPORTS SHADOW RECORDING
F2.POL=100              ;SYSTEM SUPPORTS SECONDARY POOLS
F2.WND=200              ;SYSTEM SUPPORTS SECONDARY POOL FILE WINDOWS
F2.DPR=400              ;SYSTEM HAS A SEPARATE DIRECTIVE PARTITION
F2.IRR=1000             ;INSTALL, RUN, AND REMOVE SUPPORT
F2.GGF=2000             ;GROUP GLOBAL EVENT FLAG SUPPORT
F2.RAS=4000             ;RECEIVE/SEND DATA PACKET SUPPORT
F2.AHR=10000            ;ALT. HEADER REFRESH AREA SUPPORT
F2.RBN=20000            ;ROUND ROBIN SCHEDULING SUPPORT
F2.SWP=40000            ;EXECUTIVE LEVEL DISK SWAPPING SUPPORT
F2.STP=100000           ;EVENT FLAG MASK IS IN THE TCB(1=YES)


;
; THIRD FEATURE MASK SYMBOL DEFINITIONS
;
F3.CRA=1                ;SYSTEM SPONTANEOUSLY CRASHED (1=YES)
F3.XCR=2                ;SYSTEM CRASHED FROM XDT (1=YES)
F3.EIS=4                ;SYSTEM REQUIRES EXTENDED INSTRUCTION SET
F3.STM=10               ;SYSTEM HAS SET SYSTEM TIME DIRECTIVE
F3.UDS=20               ;SYSTEM SUPPORTS USER DATA SPACE
F3.PRO=40               ;SYSTEM SUPPORTS SEC. POOL PROTO TCBS
F3.XHR=100              ;SYSTEM SUPPORTS EXTERNAL TASK HEADERS
F3.AST=200              ;SYSTEM HAS AST SUPPORT
F3.11S=400              ;RSX-11S SYSTEM
F3.CLI=1000             ;MULTIPLE CLI SUPPORT
F3.TCM=2000             ;SYSTEM HAS SEPARATE TERMINAL DRIVER POOL
F3.PMN=4000             ;SYSTEM SUPPORTS POOL MONITORING
F3.WAT=10000            ;SYSTEM HAS WATCHDOG TIMER SUPPORT
F3.RLK=20000            ;SYSTEM SUPPORTS RMS RECORD LOCKING
F3.SHF=40000            ;SYSTEM SUPPORTS SHUFFLER TASK


;
; FOURTH FEATURE MASK BITS
;
F4.CXD=1                ;COMM EXEC IS DEALLOCATED (NON-I/D ONLY)


;
; HARDWARE FEATURE MASK BIT DEFINITIONS
;
;       HF.CIS,HF.FPP DEFINED AS SIGN BITS FOR RUN TIME SPEED
;
HF.UBM=1                ;PROCESSOR HAS A UNIBUS MAP (1=YES)
```

HWDDF$ (Cont.)

```
HF.EIS=2                ;PROCESSOR HAS EXTENDED INSTRUCION SET
HF.CIS=200              ;PROCESSOR SUPPORTS COMMERCIAL INST SET
HF.FPP=100000           ;(1=PROC. HAS NO FLOATING POINT UNIT)


;
; 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.
;
SF.STD=1                ;STANDARD EXEC SELECTED
SF.RL2=2                ;SYSTEM IS FROM RL02 KIT


;
; MULTIPROCESSOR STATUS TABLE DEFINITIONS (TEMPORARY)
;
MP.CRH=100000           ;CRASH PROCESSOR IMMEDIATELY
MP.PWF=40000            ;POWERFAIL ON ONE CPU
MP.RSM=20000            ;RESET INTERRUPT MASKS
MP.NOP=10000            ;NOP FUNCTION FOR TRANSMISSION CHECK
MP.STP=4                ;STOP PROCESSOR IN ORDERLY FASHION
MP.INT=7777             ;BIC MASK FOR INTERRUPT LVL FUNCTIONS
```

ITBDF$

```
                ITBDF$  ,,SYSDEF

        ;
        ; INTERRUPT TRANSFER BLOCK (ITB) OFFSET DEFINITIONS
        ;

                .MCALL  PKTDF$
                PKTDF$          ; DEFINE AST BLOCK OFFSETS


                .ASECT
        .=0
000000  X.LNK:  .BLKW   1       ; LINK WORD FOR ITB LIST STARTING IN TCB
000002  X.JSR:  JSR     R5,@#0  ; CALL $INTSC
000006  X.PSW:  .BLKB   1       ; LOW BYTE OF PSW FOR ISR
000007          .BLKB   1       ; UNUSED
000010  X.ISR:  .BLKW   1       ; ISR ENTRY POINT (APR5 MAPPING)
000012  X.FORK:                 ; FORK BLOCK
000012          .BLKW   1       ; THREAD WORD
000014          .BLKW   1       ; FORK PC
000016          .BLKW   1       ; SAVED R5
000020          .BLKW   1       ; SAVED R4
000022  X.REL:  .BLKW   1       ; RELOCATION BASE FOR APR5
000024  X.DSI:  .BLKW   1       ; ADDRESS OF DIS.INT. ROUTINE
000026  X.TCB:  .BLKW   1       ; TCB ADDRESS OF OWNING TASK

                .IF NB  SYSDEF

000030          .BLKW   1       ; A.DQSR FOR AST BLOCK
000032  X.AST:  .BLKB   A.PRM   ; AST BLOCK
000044  X.VEC:  .BLKW   1       ; VECTOR ADDRESS (IF AST SUPPORT,
                                ; THIS IS FIRST AND ONLY AST PARAMETER)
000046  X.VPC:  .BLKW   1       ; SAVED VECTOR PC
000050  X.LEN:                  ; LENGTH IN BYTES OF ITB

                .ENDC

                .PSECT
```

KRBDF$

```
                KRBDF$

        ;
        ; CONTROLLER REQUEST BLOCK (KRB)
        ;
        ; THE CONTROLLER REQUEST BLOCK DEFINES THE ENVIRONMENT OF A DEVICE
        ; CONTROLLER.  EXACTLY ONE KRB EXISTS FOR EVERY DEVICE CONTROLLER
        ; IN AN RSX-11M+ SYSTEM.  THE KRB CONTAINS CERTAIN DEVICE STATUS
        ; INCLUDING THE CSR AND VECTOR ADDRESS FOR THE CONTROLLER.
        ;
                .ASECT
        .=177770
177770  K.PRM:  .BLKW   1       ;DEVICE DEPENDANT PARAMETER WORD
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       ;CONTROLLER UNIBUS RUN MASK
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.
        ;


        ;
        ; DEFINE OFFSETS IN SCB/KRB FOR DISK MSCP CONTROLLERS
        ;
        .=-20.
177754  KE.UMH: .BLKW   2       ;LIST HEAD FOR UMR WAITING ASSIGNMENT BLK(S)
177760  KE.UMC: .BLKW   1       ;COUNT OF AVAILABLE UMR WAITING ASSIGNMENT
                                ;BLOCK(S)


        .=177776
177776  KE.RHB: .BLKW   1       ;OFFSET TO RHBAE REGISTER (IF ANY)
        ;
        ; WHEN ONE ADDS (K.OFF) TO THE KRB ADDRESS, IT YIELDS AN ADDRESS
        ; WHICH POINTS TO HERE.
        ;
000000  KE.UCB: .BLKW   1       ;OFFSET TO UCB TABLE (IF KS.UCB SET)

                .PSECT


        ;
        ; CONTROLLER REQUEST BLOCK (KRB) STATUS BIT DEFINITIONS
        ;
        KS.OFL=1                ;CONTROLLER OFFLINE (1=YES)
        KS.MOF=2                ;CONTROLLER MARKED FOR OFFLINE (1=YES)
```

KRBDF$ (Cont.)

```
        KS.UOP=4                ;SUPPORTS OVERLAPPED OPERATION (1=YES)
        KS.MBC=10               ;DEVICE IS MASSBUS CONTROLLER (1=YES)
        KS.SDX=20               ;SEEKS ALLOWED DURING DATA XFERS (1=YES)
        KS.POE=40               ;PARALLEL OPERATION ENABLED (1=YES)
        KS.UCB=100              ;UCB TABLE PRESENT (1=YES)
        KS.DIP=200              ;DATA TRANSFER IN PROGRESS (1=YES)
        KS.PDF=400              ;PRIVILEGED DIAGNOSTIC FUNCTIONS ONLY(1=YES)
        KS.EXT=1000             ;EXTENDED 22-BIT UNIBUS CONTROLLER (1=YES)
        KS.SLO=2000             ;CONTROLLER IS SLOW COMING ONLINE (1=YES)


        ;
        ; DEFINE THE CONTIGUOUS SCB OFFSETS
        ;
                .ASECT
        .=177762
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
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
```

LCBDF$

```
                LCBDF$

        ;
        ; LOGICAL ASSIGNMENT CONTROL BLOCK
        ;
        ; THE LOGICAL ASSIGNMENT CONTROL BLOCK (LCB) IS USED TO ASSOCIATE A
        ; LOGICAL NAME WITH A PHYSICAL DEVICE UNIT.  LCB'S ARE LINKED
        ; TOGETHER TO FORM THE LOGICAL ASSIGNMENTS OF A SYSTEM.  ASSIGNMENTS
        ; MAY BE ON A SYSTEM WIDE OR LOCAL (TERMINAL) BASIS.
        ;
                .ASECT
        .=0
000000  L.LNK:  .BLKW   1       ;LINK TO NEXT LCB
000002  L.NAM:  .BLKW   1       ;LOGICAL NAME OF DEVICE
000004  L.UNIT: .BLKB   1       ;LOGICAL UNIT NUMBER
000005  L.TYPE: .BLKB   1       ;TYPE OF ENTRY (0=SYSTEM WIDE)
000006  L.UCB:  .BLKW   1       ;TI UCB ADDRESS
000010  L.ASG:  .BLKW   1       ;ASSIGNMENT UCB ADDRESS
000012  L.LGTH=.-L.LNK          ;LENGTH OF LCB

                .PSECT
```

MTADF$

```
                MTADF$

        ;
        ; ANSI MAGTAPE SPECIFIC DATA STRUCTURES
        ;
        ; VOLUME SET CONTROL BLOCK OFFSET DEFININTIONS (VSCB)
        ;
        ; VOLUME SET AND PROCESS CONTROL SECTION
        ;
                .ASECT
        .=0
000000  V.TCNT: .BLKW   1       ;TRANSACTION COUNT
000002  V.TYPE: .BLKB   1       ;VOLUME TYPE DESCRIPTOR
000003  V.VCHA: .BLKB   1       ;VOLUME CHARACTERISTICS
000004  V.LABL: .BLKB   12.     ;FILE SET ID (FIRST SIX BYTES)
000020  V.NXT:  .BLKW   1       ;PTR TO NEXT VSCB NODE
000022  V.MVL:  .BLKW   1       ;PTR TO MOUNTED VOL LIST
000024  V.UVL:  .BLKW   1       ;PTR TO UNMOUNTED VOL LIST
000026  V.ATL:  .BLKW   1       ;ATL ADDR OF ACCESSING TASK TCB IN RSX11M
000030  V.UCB:  .BLKW   1       ;ADDR OF CURRENT UCB OR PUD
000032  V.RVOL: .BLKB   1       ;CURRENT RELATIVE VOL #
000033  V.MOU:  .BLKB   1       ;MOUNT MODE BYTE
000034  V.TCHR: .BLKW   1       ;UINT CHAR. FOR ALL UNITS USED FOR VOL SET
000036  V.SEQN: .BLKW   1       ;CURRENT FILE SEQUENCE #
000040  V.SECN: .BLKW   1       ;CURRENT FILE SECTION #
000042  V.TPOS: .BLKB   1       ;POSITION OF TAPE IN TM'S TO NXT HDR1
000043  V.PSTA: .BLKB   1       ;PROCESS STATUS BYTE
000044  V.TIMO: .BLKW   1       ;BLOCKED PROCESS TIMEOUT COUNTER
000046  V.STAT: .BLKW   3       ;STATUS WORDS USED BY COMMAND EXECUTION MODS
000054  V.TRTB: .BLKB   1       ;TRANSLATION CONTROL BYTE
000055  V.EFTV: .BLKB   1       ;FOR MAG TO RETURN IE.EOF, EOT, EOV


        ;
        ; LABEL DATA SECTION
        ;
000056  V.BLKL: .BLKW   1       ;BLOCK LENGTH
000060  V.RECL: .BLKW   1       ;RECORD LENGTH
000062  V.FNAM: .BLKW   3       ;FILE NAME
000070  V.FTYP: .BLKW   1       ;FILE TYPE
000072  V.FVER: .BLKW   1       ;FILE VERSION #
000074  V.CDAT: .BLKW   2       ;CREATION DATE
000100  V.EDAT: .BLKW   2       ;EXPRIATION DATE
000104  V.BLKC: .BLKW   2       ;BLOCK COUNT FOR FILE SECTION
000110  V.RTYP: .BLKB   1       ;RECORD TYPE
000111  V.FATT: .BLKB   1       ;FILE ATTRIBUTES FOR CARRIAGE CONTROL
000112          .BLKB   30.     ;REMAINDER OF FILE ATTRIBUTES


        ;
        ; NULL WINDOW SECTION
        ;
000150  V.WIND: .BLKW   4.      ;NULL WINDOW
000160  V.MST2: .BLKW   1       ;MAGTAPE STATUS BITS
000162  V.FABY: .BLKB   1       ;FILE ACCESSIBILITY BYTE (HDR1)
000163          .BLKB   1       ;SPARE
000164  V.ANSN: .BLKB   17.     ;ANSI 17 CHARACTER FILE NAME
000205  V.BOFF: .BLKB   1.      ;BUFFER OFFSET
000206  V.DENS: .BLKB   1.      ;REQUESTED UNIT DENSITY
000207  V.DRAT: .BLKB   1.      ;DEFAULT RECORD ATTRIBUTES
```

MTADF$ (Cont.)

```
000210  V.DBLK: .BLKW   1.       ;DEFAULT BLOCK SIZE
000212  V.DREC: .BLKW   1.       ;DEFAULT RECORD SIZE

000214  S.VSCB=.                 ;SIZE OF VSCB

                .PSECT

        ;
        ; DEFINE OFFSETS INTO NULL WINDOW SECTION
        ;
                .ASECT
        .=0
000000  W.CTL:  .BLKW   1        ;CONTROL WORD IN WINDOW
        V.WINC=V.WIND+W.CTL      ;CNTRL WORD IN NULL WINDOW
                                 ;RELATIVE TO THE VSCB

                .PSECT

        ;
        ; MOUNTED VOLUME LIST OFFSET DEFININTIONS (MVL)
        ;
                .ASECT
        .=0
000000  M.NXT:  .BLKW   1        ;PTR TO NXT MVL NODE (11M)
000002  M.UIC:  .BLKW   1        ;OWNER UIC FROM RVOL #1
000004  M.CH:   .BLKW   1        ;U.CH/U.VP (11D)
000006  M.PROT: .BLKW   1        ;PROTECTION U.AR IN 11D
000010  M.RVOL: .BLKB   1        ;RELATIVE VOL # OF MOUNTED VOLUME
000011  M.STAT: .BLKB   1        ;VOLUME STATUS
000012  M.VIDP: .BLKW   1        ;VOLUME ID POINTER
000014  M.UCB:  .BLKW   1        ;ADDR OF ASSOC UCB OR PUD

000016  S.MVL=.                  ;SIZE OF MVL NODE

                .PSECT

        ;
        ; UNMOUNTED VOLUME AND VOLUME  LIST OFFSET DEFINITIONS (UVL)
        ;
                .ASECT
        .=0
000000  L.NXT:  .BLKW   1        ;PTR TO NXT UVL NODE
000002  L.VOL1: .BLKB   1        ;REL VOL # OF 1'ST VOL IN NODE
000003  L.VOL2: .BLKB   1        ;REL VOL # OF 2'ND VOL IN NODE
000004  L.VID1: .BLKB   6        ;VOL ID OF 1'ST VOL IN NODE
000012  L.VID2: .BLKB   6        ;VOL ID OF 2'ND VOL IN NODE

000020  S.UVL=.                  ;SIZE OF UVL NODE

                .PSECT

        ;
        ; SYSTEM DATA STRUCTURE CONTENT VALUES
        ;

        ;
        ; VSCB VALUES
        ;
        ; V.MOU VALUES
        ;
        VM.OLD  =       200      ;OLD .FL300 VOLUME - VM.BYP WILL ALSO BE SET
```

MTADF$ (Cont.)

```
VM.BYP  =       100     ;BYPASS LABEL PROCESSING
VM.ULB  =       40      ;UNLABELED TAPE
VM.FSC  =       20      ;OVERRIDE FILE SET ID CHECK
VM.EXC  =       10      ;OVERRIDE EXPRIATION DATE CHECK


;
; V.MST2 VALUES
;
V2.INI  =       1       ;MAG WANTS US TO INITIALIZE NEXT OUTPUT
V2.XH2  =       2       ;THIS FILE HAS NO HDR2, DON'T WRITE EOF2
V2.XH3  =       4       ;THIS FILE HAS NO HDR3, DON'T WRITE EOF3
V2.NH3  =       10      ;DON'T WRITE HDR3/EOX3 LABELS
V2.OAC  =       20      ;OVERRIDE FILE/VOLUME ACCESSIBILITY


;
; V.PSTA VALUES - UNBLOCKED TRANSITION STATE
;
VP.RM   =       2       ;READ DATA MODE
VP.WM   =       4       ;WRITE DATA MODE
VP.UCM  =       6       ;UNLABELLED CREATE POSITIONING MODE
VP.SM   =       10      ;SEARCH MODE
VP.MOU  =       20      ;MOUNT MODE
VP.RWD  =       40      ;REWIND OR VOL VERIFICATION WAIT
VP.VFY  =       VP.RWD
VP.POS  =       100     ;PROCESS IN POSITIONING MODE
                        ;(MULTI-SECTION FILE)


;
; BLOCKED STATE = -(UNBLOCKED TRANSITION STATE VALUES)
;
; PROCESS TIMED OUT BIT 0 = 1
;
VP.TO=1


;
; NULL WINDOW CONTROL BIT DEFINITIONS
;
WI.RDV  =       400     ;ACCESSED FOR READ
WI.WRV  =       1000    ;ACCESSED FOR WRITE
WI.EXT  =       2000    ;ACCESSED FOR EXTEND
WI.LCK  =       4000    ;LOCKED


;
; MVL VALUES IN THE M.STAT FIELD
;
MS.VER  =       200     ;VOL ID NOT VERIFIED
MS.RID  =       1       ;VOL ID TO BE READ NOT CHECKED
MS.NMO  =       2       ;MOUNT MESSAGE NOT GIVEN YET
MS.TMO  =       4       ;ONE TIMEOUT ALREADY EXPRIED
MS.EXP  =       10      ;EXPIRATION DATE MESSAGE GIVEN
```

MTADF$ (Cont.)

```
;
; MISC BITS USED IN MOUNT (STORED IN V.STS)
;
MO.OVR  =       1       ;OVER RIDE VOL NAME SWITCH
MO.UIC  =       2       ;EXPLICIT UIC GIVEN
MO.PRO  =       4       ;EXPLICIT PROTECTION GIVEN
MO.160  =       10      ;1600 BPI SPECIFIED
```

OLRDF$

```
        OLRDF$  $$$GBL

;
; THIS MODULE DEFINES THE ONLINE RECONFIGURATION INTERFACE
; AS IMPLEMENTED BETWEEN THE RSX-11M-PLUS TASKS CON, HRC, AND
; THE RDDRV.
;


;
; DEFINE THE I/O FUNCTION CODES FOR ONLINE RECONFIGURATION CONTROL.
;
        .MCALL  .WORD.,DEFIN$

        .IF IDN <$$$GBL>,<DEF$G>
...GBL=1
        .IFF
...GBL=0
        .ENDC


;
; THE FOLLOWING MACRO DEFINES THE SUB-FUNCTION CODES FOR EACH OF THE
; OPERATIONS PERFORMED BY THE HRC TASK AND A PARAMETER DESCRIBING
; THE ARGUMENTS REQUIRED FOR EACH FUNCTION.  IN A MACRO CALL THE
; FOLLOWING ARE THE LEGAL COMBINATIONS FOR THE 'MASK'
; PARAMETER:
;               <>        SIGNIFYING NO PARAMETERS
;               <D>       SIGNIFYING ONE BUFFER DESCRIPTOR
;               <D,D>     SIGNIFYING TWO BUFFER DESCRIPTORS
;               <D,CT>    SIGNIFYING ONE DESCRIPTOR AND 'CT' BYTES OF
;                         PARAMETERS
;               <CT>      SIGNIFYING 'CT' BYTES OF PARAMETERS
;

        .MACRO  FUNC NAME,SUBF,FUN,MASK
        .WORD.  IO.'NAME,SUBF,FUN
        FUNCA   NAME,<MASK>
        .ENDM

        .MACRO  FUNCA NAME,MSK
        PARCT=0
        DESCT=0
        .IRP X,<MSK>
        .IIF IDN <X>,<P> PARCT=PARCT+1
        .IIF IDN <X>,<D> DESCT=DESCT+1
        .IIF GT <PARCT-17> .ERROR INVALID PARAMETER COUNT
        .IIF GT <DESCT-17> .ERROR INVALID DESCRIPTOR COUNT
        .ENDR

        TEMP=<DESCT*4>+<PARCT*2>
        .WORD.  IO$'NAME,<<DESCT*20+PARCT>>,TEMP
        .ENDM

;
; DEFINE ONLINE RECONFIGURATION I/O FUNCTIONS
;

        .WORD.  IO.MFC,000,001  ; MULTI-FUNCTION MODIFY CONFIGURATN
        .WORD.  IO.RSC,000,002  ; READ SYSTEM CONFIGURATION
        .WORD.  IO.WSC,000,006  ; MODIFY DEVICE CONFIGURATION
```

OLRDF$ (Cont.)

```
;
; DEFINE SUBFUNCTIONS TO MODIFY DEVICE CONFIGURATION
;

        FUNC    ONL,001,006,<D,D>        ; SET DEVICE ONLINE
        FUNC    OFL,002,006,<D,D>        ; SET DEVICE OFFLINE
        FUNC    MAI,003,006,<D,D>        ; SET DEVICE IN MAINT MODE
        FUNC    CAC,004,006,<>           ; CACHE CONTROL
        FUNC    MEM,005,006,<>           ; MIND CONTROL
        FUNC    STN,006,006,<P,P>        ; RECONFIGURATION CONTROL,
                                         ; SPECIFY TASK NAME
        FUNC    HRC,007,006,<P,P>        ; RECONFIGURATION CONTROL,
                                         ; HRC OPERATING MODE
        FUNC    ONE,010,006,<P,P>        ; ON <CONDITION> <COMMAND>
        FUNC    STA,011,006,<D>          ; RETURN DEVICE STATE
        FUNC    IF ,012,006,<P,P>        ; IF <CONDITION> <COMMAND>
        FUNC    RLI,013,006,<D,D,D,D>    ; LINK UNIBUS RUN
        FUNC    RUL,014,006,<D,D,D,D>    ; UNLINK UNIBUS RUN
        FUNC    MBO,015,006,<P,P,D,D,D,D,D,D,D,D> ; MEM BOX ONLINE
        FUNC    RSW,016,006,<D,D,D,D>    ; SWITCH BUS
        FUNC    WAT,017,006,<D>          ; WRITE ATTRIBUTES
        FUNC    RAT,020,006,<D,D>        ; READ ATTRIBUTES
        FUNC    MBF,021,006,<P,P,D,D,D,D,D,D,D,D> ; MEM BOX OFFLINE

        IO$MAX=21                ; DEFINE MAXIMUM SUBFUNCTION


        DEFIN$  IS.HRG,6.        ; STOP PROCESSING COND ENCOUNTERED
                                 ; SECOND STATUS WORD IS ARGUMENT


;
; DEFINE A MACRO, WHICH WHEN EXPANDED WITH THE APPROPRIATE
; DEFINITION FOR .IOER. WILL DEFINE THE PRIVATE ERROR CODES USED BY
; HRC AND CON.
;

.MACRO  OLREM$

$$$VAL=-256.                     ; DEFINE INITIAL ERROR NUMBER VALUE

.IOER.  IE$DAL,<DEVICE already linked>
.IOER.  IE$DNL,<DEVICE not linked>
.IOER.  IE$PRM,<Parameter error>
.IOER.  IE$SYN,<Syntax error>
.IOER.  IE$AFE,<Attribute format error>
.IOER.  IE$TMU,<HRC... Internal tables insufficient for this system>
.IOER.  IE$CAB,<Unable to access busrun>
.IOER.  IE$TRP,<HRC... internal addressing error>
.IOER.  IE$ALG,<Memory box parameter error>
.IOER.  IE$TQU,<Timeout on unit quieting operation>
.IOER.  IE$EPO,<ONLINE CPU failure>
.IOER.  IE$EUO,<ONLINE UNIT failure>
.IOER.  IE$ECO,<ONLINE CONTROLLER failure>
.IOER.  IE$EPF,<OFFLINE CPU failure>
.IOER.  IE$EUF,<OFFLINE UNIT failure>
.IOER.  IE$ECF,<OFFLINE CONTROLLER failure>
.IOER.  IE$CFU,<Attempt to quiet unit for controller failed>
.IOER.  IE$CSR,<CSR for controller not present in I/O page>
```

OLRDF$ (Cont.)

```
        .IOER.  IE$SWF,<Unable to switch unit away from current controller>
        .IOER.  IE$ICE,<HRC... detected I/O database consistancy error>
        .IOER.  IE$SCE,<Executive or Driver status change error>
        .IOER.  IE$MDE,<HRC... Memory descriptor format error>
        .IOER.  IE$NFW,<No path to target device is available>
        .IOER.  IE$CXT,<Unable to take unit with context offline.>
        .IOER.  IE$IDU,<Invalid device descriptor>
        .IOER.  IE$UNK,<Device is unknown in this configuration>
        .IOER.  IE$SZE,<HRC... Unable to access device to size drive>
        .IOER.  IE$POB,<HRC... Can't take box offline. Partition overmaps box>
        .IOER.  IE$NLB,<HRC... Can't take box offline. Not last box in memory>
        .IOER.  IE$OMP,<HRC... Can't modify partition size. Overmap exists>
        .IOER.  IE$POC,<HRC... Can't modify partition size. Occupied>
        .IOER.  IE$DFE,<HRC... Request format error.>
        .IOER.  IE$IDS,<HRC... Invalid device specification.>
        .IOER.  IE$UOE,<HRC... Unkown error from online/offline call>
        .ENDM


        ;
        ; CONDITION CODES FOR CONDITIONS TESTED BY IO.ONE AND IO.IF FUNCTS
        ;
                CO$ONL = 1      ; IF DEVICE NOW ONLINE
                CO$OFL = 2      ; IF DEVICE NOW OFFLINE
                CO$UNK = 3      ; UNKNOWN DEVICE
                CO$ACC = 4      ; ACCESSABLE (ACCESS PATH EXISTS)
                CO$ANY = 5      ; ANY ERROR CONDITION
                CO$MAI = 6      ; MAINTENANCE MODE

                CO$MAX = 6      ; MAXIMUM CODE


        ;
        ; CONDITION COMMAND CODES FOR IO.ONE AND IO.IF FUNCTIONS
        ;
                CD$STO = 2      ; 'STOP' COMMAND
                CD$GOT = 4      ; 'GOTO'
                CD$CON = 6      ; 'CONTINUE'

                CD$MAX = 6      ; MAXIMUM CONDITION DEFINED


        ;
        ; ARGUMENT DEFINITION FOR IO.HRC FUNCTION
        ;
                M$LOG = 1       ; SUPRESS CONFIG TRANSMISSION TO ERRLOG
                M$INIT = 2      ; INITALIZE HRC
                M$DEBG = 4      ; SET HRC INTO DEBUG MODE (DEVELOPMENT ONLY)
                M$EXIT = 10     ; EXIT REQUEST (FROM ABORT AST REQUEST)


        ;
        ; DEFINE TABLE OFFSETS AND STATUS BITS RETURNED IN RESPONSE TO
        ; A 'READ CONFIGURATION' QIO
        ;
                .ASECT
        .=0
000000  C$DTYP: .BLKB   1       ; ENTRY TYPE FIELD
```

OLRDF$ (Cont.)

```
        ;
        ; ENTRY TYPE CODES ARE AS FOLLOWS
        ;
                ET$HDR = 1      ; CONFIGURATION HEADER ENTRY
                ET$END = 2      ; END OF CONFIGURATION DATA

                ET$DEV = 'A     ; MIN VALUE FOR DEVICE SPECIFICATION ENTRY


000001  C$DECT: .BLKB   1       ; COUNT OF TABLE ENTRIES (CPUS+SWITCHED
                                ; BUS RUNS+CONTROLLERS+UNITS)
000002  C$DVER: .BLKB   1       ; VERSION OF RECONFIGURATION TASK PROTOCAL
000003  C$DSTD: .BLKB   1       ; SIZE OF HEADER
000004  C$DMUB: .BLKB   1       ; MAXIMUM UNIBUS RUNS SUPPORTED
000005  C$DMCT: .BLKB   1       ; MAX CONTROLLERS OF A GIVEN TYPE SUPPORTED
                .EVEN
000006  C$DFAC: .BLKW   2       ; FACILITES SUPPORTED IN HOST SYSTEM
000012  C$DIDN: .BLKW   9.      ; HRC VERSION AND BUILD TIMESTAMP

000034  C$STD:                  ; SIZE OF THE TABLE HEADER


        ;
        ; OFFSETS WITHIN THE FIXED PORTION OF A GIVEN ENTRY
        ;
        .=0
000000  C$DTYP:                 ; ENTRY TYPE CODE
000000  C$DNAM: .BLKW   1       ; TWO ASCII CHARACTER UNIT OR CONTR NAME
000002  C$DPUN: .BLKB   1       ; CONTROLLER NUMBER (0-255.)
000003  C$DLUN: .BLKB   1       ; LOGICAL UNIT NUM IF THIS DEVICE IS A UNIT
000004  C$DSCT: .BLKB   1       ; SUB-CONTROLLER NUMBER
000005  C$DEVT: .BLKB   1       ; DEVICE TYPE CODE
000006  C$DSTS: .BLKW   1       ; DEVICE STATUS MASK


        ;
        ; FLAG VALUES FOR C$DSTS
        ;
                CS$ATR=1        ; VARIABLE LENGTH ATTRIBUTE INFO IS APPENDED
                CS$EXF=76       ; FIELD IN C$DSTS CONTAINING COUNT OF
                                ; ADDITIONAL BYTES IN THIS DEVICE ENTRY
                CS$SUB=100      ; THIS IS A SUB-CONTROLLER DEVICE
                ;CS$XXX=200     ; UNUSED
                CS$OFL=400      ; 1=>DEVICE IS OFFLINE, 0=>DEVICE IS ONLINE
                CS$PDF=1000     ; DEV IS RESTRICTED TO PRIVILEGED DIAG FNS
                CS$POR=2000     ; THIS IS A MULTIPORT DEVICE
                CS$MBD=4000     ; DEVICE IS A MASS BUS DEVICE
                CS$UNK=10000    ; DEVICE IS UNKNOWN
                CS$ACC=20000    ; AN ONLINE ACCESS PATH EXISTS TO THIS DEV
                CS$MTD=40000    ; DEV IS MOUNTED(DISK) OR LOGGED IN (TERM)
                CS$DRV=100000   ; A DRIVER IS LOADED FOR THIS DEVICE


000010  C$DST2: .BLKW   1       ; STATUS EXTENSION


                CS$PUN=20       ; 1=> THIS DEVICE SPECIFIED WITH PHYSICAL
                                ;     UNIT NUMBER
                CS$CRD=40       ; 1=> THIS IS A CONTROLLER RELATIVE DEVICE
                                ;     SPEC
```

OLRDF$ (Cont.)

```
                CS$PRC=100        ; 1=> THIS IS A PORT RELATIVE CONTROLLER
                                  ;     SPEC
                CS$CTL=200        ; DEVICE IS A CONTROLLER (MUST BE SIGN BIT)
                CS$DCL=3400       ; DEVICE CLASS CODE FIELD. MUST BE LOW ORDER
                                  ; BITS OF HIGH BYTE.


        ;
        ; DEVICE CLASS VALUES
        ;
                DC$UNI = 0        ; UNIT
                DC$CTL = 1        ; CONTROLLER
                DC$MKU = 2        ; MEMORY BOX UNIT
                DC$MKC = 3        ; MEMORY BOX CONTROLLER
                DC$SBU = 4        ; SWITCHED BUS UNIT
                DC$SBC = 5        ; SWITCHED BUS CONTROLLER
                DC$CPU = 6        ; CPU
                ;DC$XXX = 7       ; UNUSED

000012  C$DDAT: .BLKW   2         ; DEVICE DEPENDANT DATA

000016  C$SME:                    ; SIZE IF A MINIMUM ENTRY


        ;
        ; VARIABLE PORTION OF A GIVEN ENTRY
        ;

        ;
        ; FOR CONTROLLERS
        ;
        .=C$SME
000016  C$DKPO: .BLKW   1         ; PORT-STATUS-WORD. THIS DESCRIBES THE BUS
                                  ; RUN, CPU OR SWITCHED BUS, TO WHICH THIS
                                  ; CONTROLLER IS CONNECTED.
000020  C$SCT:                    ; MIMIMUM SIZE OF A CONTROLLER ENTRY


        ;
        ; FOR UNIT ENTRIES
        ;
        .=C$SME
000016  C$DCTN: .BLKW   1         ; CONTROLLER NAME. TWO CHARACTER ASCII CODE
                                  ; OF THE CONTROLLER TO WHICH THIS UNIT IS
                                  ; ATTACHED.
000020  C$DUPO: .BLKW   1         ; PORT-STATUS-WORD. THIS IS THE
                                  ; FIRST OF THE PSWS DESCRIBING THE CONTR(S)
                                  ; TO WHICH THIS UNIT IS CONNECTED.
000022  C$SUN:                    ; MIMIMUM SIZE OF A UNIT ENTRY


        ;
        ; FOR CPU-S
        ;
        .=C$SME
000016  C$DCPO: .BLKW   1         ; PORT-STATUS-WORD. THIS IS THE BUS
                                  ; NUMBER FOR THIS CPU.
000020  C$SCP:                    ; MINIMUM SIZE OF A CPU ENTRY
```

OLRDF$ (Cont.)

```
        ;
        ; FOR MEMORY BOXES
        ;
        .=C$SME
000016  C$DCTN: .BLKW   1          ; CONTROLLER NAME.
000020          .BLKW   4          ; MAXIMUM OF 4 PORTS FOR MEMORY CONTROLLERS

000030  C$SMB:                     ; MAXIMUM SIZE OF A MEMORY BOX ENTRY


        ;
        ; STATUS BIT DEFINITIONS FOR THE PORT STATUS WORD
        ;
                CP$OFL=400         ; 1=> PORT IS OFFLINE
                CP$XXX=1000        ; UNUSED
                CP$CUR=2000        ; THIS PORT IS THE CURRENT PORT (S.KRB
                                   ; REFERENCES THIS PORT
                CP$XXX=4000        ; UNUSED
                CP$XXX=10000       ; UNUSED
                CP$ACC=20000       ; THIS PORT HAS AN ACCESS PATH
                CP$MTD=40000       ; PORT HAS CONTEXT OR SERVICES A DEVICE
                                   ; HAVING CONTEXT
                CP$XXX=100000      ; UNUSED


        ;
        ; DEVICE ATTRIBUTES CODES
        ;
                .MACRO ATT NAME,SIZ
                $$$TMP=$$$TMP+1
                DEFIN$ DA$'NAME,$$$TMP!<400*SIZ>
                .ENDM

                $$$TMP=0

                ATT     CSR,2      ; CSR ADDRESS
                ATT     VEC,2      ; VECTOR ADDRESS
                ATT     UBR,2      ; UNIBUS RUN
                ATT     TYP,2      ; DEVICE TYPE, READ ONLY
                ATT     VOL,12.    ; MOUNTED VOLUME NAME, READ ONLY
                ATT     ERR,10     ; DEVICE ERROR COUNTERS, READ/WRITE
                ATT     PRI,2      ; DEVICE INTERRUPT PRIORITY
                ATT     MBP,6      ; MEMORY BOX PARAMETER
                ATT     STE,2      ; SANITY TIMER ENABLE/DISABLE
                ATT     SAL,2      ; ALARM ENABLE/DISABLE
                ATT     DSN,2      ; DEVICE SERIAL NUMBER
                ATT     CSN,10     ; CPU SERIAL NUMBERS


        ;
        ; MEMORY BOX ATTRIBUTE BUFFER
        ;
                .ASECT
        .=0
000000  C$MBAS: .BLKW   1          ; BASE ADDRESS OF BOX
000002  C$MINT: .BLKB   1          ; INTERLEAVE FACTOR
000003          .BLKB   1          ; FREE BYTE
000004  C$MSIZ: .BLKW   1          ; SIZE OF BOX IN 32 WORD BLOCKS
000006  C$MGRN: .BLKW   1          ; BOX GRANULARITY. "BYTES-PER-UNIT"
```

OLRDF$ (Cont.)

```
000010  C$MDSC:                 ; SIZE OF BOX ATTRIBUTE BUFFER

                .PSECT


        ;
        ; MACRO FOR THE DEFINITION OF DEVICE TYPE CODES
        ;
                .MACRO DEVCD$ $$$GBL

                .MCALL DEFIN$

                .IF IDN <$$$GBL>,<DEF$G>
        ...GBL=1
                .IFF
        ...GBL=0
                .ENDC

                .MACRO DEV X
                DEFIN$ D$'X,$$$TMP
                $$$TMP=$$$TMP+1
                .ENDM

                $$$TMP = 0

                DEV UDET        ; UNDETERMINED DEVICE TYPE
                DEV UKNO        ; UNKNOWN DEVICE TYPE

                DEV RK03        ; RK03
                DEV RK05        ; RK05
                DEV RK5F        ; RK05-F (DUAL DENSITY FIXED CARTRIDGE)

                DEV RX01        ; RX01
                DEV RX02        ; RX02 (DUAL DENSITY RX01)

                DEV RL01        ; RL01
                DEV RL02        ; RL02

                DEV RP02        ; RP02
                DEV RP03        ; RP03
                DEV RP04        ; RP04
                DEV RP05        ; RP05
                DEV RP06        ; RP06
                DEV RP07        ; RP07

                DEV RK06        ; RK06
                DEV RK07        ; RK07

                DEV RM02        ; RM02
                DEV RM03        ; RM03
                DEV RM05        ; RM05
                DEV RM80        ; RM80

                DEV RS03        ; RS03
                DEV RS04        ; RS04 (DUAL DENSITY RS03)

                DEV RF11        ; RF11/RS08
```

OLRDF$ (Cont.)

```
DEV TU10          ; TU10
DEV TU16          ; TU16
DEV TU45          ; TU45
DEV TU77          ; TU77
DEV TU78          ; TU78
DEV TS11          ; TS11

DEV TM02          ; TM02
DEV TM03          ; TM03
DEV TM78          ; TM78

DEV TU56          ; TU56
DEV TU58          ; TU58
DEV TU60          ; TU60

DEV MSCP          ; UDA50
DEV RA60          ; RA60
DEV RA80          ; RA80
DEV RA81          ; RA81

DEV ML11          ; ML11

DEV TERM          ; TERMINAL

$$$TMP=370
DEV USR0          ; USER TYPE 0
DEV USR1          ; USER TYPE 1
DEV USR2          ; USER TYPE 2
DEV USR3          ; USER TYPE 3
DEV USR4          ; USER TYPE 4
DEV USR5          ; USER TYPE 5
DEV USR6          ; USER TYPE 6
DEV USR7          ; USER TYPE 7
```

PCBDF$

```
                PCBDF$  ,,SYSDEF

        ;
        ; 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 RAD50
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 PCB'S
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  SYSDEF

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
000004  P.NAM:  .BLKW   2       ;PARTITION NAME IN RAD50
000010  P.SUB:  .BLKW   1       ;POINTER TO NEXT SUBPARTITION
000012  P.MAIN: .BLKW   1       ;POINTER TO MAIN PARTITION
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          .BLKW   1       ;(UNUSED)
000022  P.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
```

PCBDF$ (Cont.)

```
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


        ;
        ; 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 RAD50
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 ADDRESS 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 EXTENSION FOR COMMON AND MAIN PCB'S
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

        $$$=.
        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)
```

PCBDF$ (Cont.)

```
        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 L
        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 B
        PR.CPA=1                 ;PROCESSOR A


        ;
        ; STATUS EXTENSION WORD BIT DEFINITIONS
        ;       (THESE BITS CAN ONLY BE EXAMINED IN COMMON OR MAIN PCB'S)
        ;
        P2.LMA=40000             ;DON'T SHUFFLE,DELETE SPINDLE OR MUTILATE
                                 ;THIS PARTITION
        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 PCB'S
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 PCB'S
```

PCBDF$ (Cont.)

```
        ;
        ; 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 ADDR 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
        ;
                .PSECT
        AS.PRO=100              ;A.TCB IS SEC POOL 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)
```

PKTDF$

```
                PKTDF$

        ;
        ; ASYNCHRONOUS SYSTEM TRAP CONTROL BLOCK OFFSET DEFINITIONS
        ;
        ; SOME POSITIONAL DEPENDENCIES BETWEEN THE OCB AND THE AST CONTROL
        ; BLOCK ARE RELIED UPON IN THE ROUTINE $FINXT IN THE MODULE SYSXT.
        ;
                .ASECT
        .=177774
177774  A.KSR5: .BLKW   1           ;SUBROUTINE KISAR5 BIAS (A.CBL=0)
177776  A.DQSR: .BLKW   1           ;DEQUEUE SUBROUTINE ADDRESS (A.CBL=0)
000000          .BLKW   1           ;AST QUEUE THREAD WORD
000002  A.CBL:  .BLKW   1           ;LENGTH OF CONTROL BLOCK IN BYTES
                                    ;IF A.CBL = 0, THE AST CONTROL BLOCK IS
                                    ;TO BE DEALLOCATED BY THE DEQUEUE SUBROUTINE
                                    ;POINTED TO BY A.DQSR MAPPED VIA APR 5
                                    ;VALUE A.KSR5.  THIS IS CURRENTLY USED ONLY
                                    ;BY THE FULL DUPLEX TERMINAL DRIVER FOR
                                    ;UNSOLICITED CHARACTER ASTS.
                                    ;IF THE LOW BYTE OF A.CBL = 0, AND THE
                                    ;HIGH BYTE IS NOT = 0, THE AST CONTROL BLOCK
                                    ;IS A 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


        ;
        ; 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
```

PKTDF$ (Cont.)

```
       ;
       ; 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


       ;
       ; BIT DEFINITIONS FOR THE GET/SET INFORMATION DIRECTIVE.
       ;
       SF.PRV=100000              ;FUNCTION IS PRIVILEGED
       SF.IN= 40000               ;FUNCTION IS AN INPUT FUNCTION


       ;
       ; GROUP GLOBAL EVENT FLAG BLOCK OFFSETS
       ;
       .=0
000000 G.LNK:  .BLKW   1          ;LINK WORD
000002 G.GRP:  .BLKB   1          ;GROUP NUMBER
000003 G.STAT: .BLKB   1          ;STATUS BYTE
000004 G.CNT:  .BLKW   1          ;ACCESS COUNT
000006 G.EFLG: .BLKW   2          ;EVENT FLAGS

000012 G.LGTH=.                   ;LENGTH OF GROUP GLOBAL EVENT FLAG BLOCK


       GS.DEL=1                   ;STATUS BIT -- MARKED FOR DELETE


       ;
       ; EXECUTIVE POOL MONITOR CONTROL FLAGS
       ;

       ;
       ; $POLST IS THE SYNCHRONIZATION WORD BETWEEN THE EXEC AND POOL
       ; MONITOR
       ;
       PC.HIH=1                   ;HIGH POOL LIMIT CROSSED (1=YES)
       PC.LOW=2                   ;LOW POOL LIMIT CROSSED (1=YES)
       PC.ALF=4                   ;POOL ALLOCATION FAILURE (1=YES)
       PC.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
       ;
       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
```

PKTDF$ (Cont.)

```
        ;
        ; 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


        ;
        ; I/O PACKET OFFSET DEFINITIONS
        ;
                 .ASECT
        .=0
000000  I.LNK:   .BLKW   1        ;I/O QUEUE THREAD WORD
000002  I.PRI:   .BLKB   1        ;REQUEST PRIORITY
000003  I.EFN:   .BLKB   1        ;EVENT FLAG NUMBER
000004  I.TCB:   .BLKW   1        ;TCB ADDRESS OF REQUESTOR
000006  I.LN2:   .BLKW   1        ;POINTER TO SECOND LUN WORD
000010  I.UCB:   .BLKW   1        ;POINTER TO UNIT CONTROL BLOCK
000012  I.FCN:   .BLKW   1        ;I/O FUNCTION CODE
000014  I.IOSB:  .BLKW   1        ;VIRTUAL ADDRESS OF I/O STATUS BLOCK
000016           .BLKW   1        ;I/O STATUS BLOCK RELOCATON BIAS
000020           .BLKW   1        ;I/O STATUS BLOCK ADDRESS
000022  I.AST:   .BLKW   1        ;AST SERVICE ROUTINE ADDRESS
000024  I.PRM:   .BLKW   1        ;RESERVED FOR MAPPING PARAMETER #1
000026           .BLKW   6        ;PARAMETERS 1 TO 6
000042           .BLKW   1        ;USER MODE DIAGNOSTIC PARAMETER WORD

000044  I.ATTL=.                  ;MINIMUM LENGTH OF I/O PACKET (USED BY
                                  ;FILE SYSTEM TO CALCULATE MAXIMUM
                                  ;NUMBER OF ATTRIBUTES)
000044  I.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


        ;
        ; 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
```

PKTDF$ (Cont.)

```
        ;
        ; 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


        ;
        ; 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


        ;
        ; 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 TT
        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


        ;
        ; 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
```

PKTDF$ (Cont.)

```
        ;
        ; ANCILLARY CONTROL BLOCK (ACB) DEFINITIONS
        ;
        .=0
000000  A.REL:   .BLKW   1        ;ACD RELOCATION BIAS
000002  A.DIS:   .BLKW   1        ;ACD DISPATCH TABLE POINTER
000004  A.MAS:   .BLKW   1        ;ACD FUNCTION MASK
000006  A.NUM:   .BLKB   1        ;ACD IDENTIFICATION NUMBER
000007           .BLKB   1        ;RESERVED
000010  A.LIN:   .BLKW   1        ;ACD LINK WORD
000012  A.ACC:   .BLKB   1        ;ACD ACCESS COUNT
000013  A.STA:   .BLKB   1        ;ACD STATUS BYTE

000014  A.LEN1=.                  ;LENGTH OF PROTOTYPE ACB


        .=A.LIN                   ;FULL ACB OVERLAPS PROTOTYPE ACB
000010  A.IMAP: .BLKW   1         ;ACD INTERRUPT BUFFER RELOCATION BIAS
000012  A.IBUF: .BLKW   1         ;ACD INTERRUPT BUFFER ADDRESS
000014  A.ILEN: .BLKW   1         ;ACD INTERRUPT BUFFER LENGTH
000016  A.SMAP: .BLKW   1         ;ACD SYSTEM STATE BUFFER RELOCATION BIAS
000020  A.SBUF: .BLKW   1         ;ACD SYSTEM STATE BUFFER ADDRESS
000022  A.SLEN: .BLKW   1         ;ACD SYSTEM STATE BUFFER LENGTH
000024  A.IOS:  .BLKW   2         ;ACD I/O STATUS
000030  A.RES:  .BLKW   1         ;RESERVED FOR USE BY THE ACD

000032  A.LEN2=.                  ;LENGTH OF FULL ACB


        ;
        ; DEFINE THE FLAG VALUES IN THE OFFSET U.AFLG
        ;
        UA.ACC=1                  ;ACCEPT THIS CHARACTER
        UA.PRO=2                  ;PROCESS THIS CHARACTER
        UA.ECH=4                  ;ECHO THIS CHARACTER
        UA.TYP=10                 ;FORCE THIS CHARACTER INTO TYPEAHEAD
        UA.SPE=20                 ;THIS CHARACTER HAS A SPECIAL ECHO
        UA.PUT=40                 ;PUT THIS CHARACTER IN THE INPUT BUFFER
        UA.CAL=100                ;CALL THE ACD BACK AFTER THE TRANSFER
        UA.COM=200                ;COMPLETE THE INPUT REQUEST
        UA.ALL=400                ;ALLOW PROCESSING OF THIS I/O REQUEST
        UA.TRA=1000               ;TRANSFER CHARS. WHEN I/O COMPLETES


        ;
        ; DEFINE THE ACD ENTRY POINTS (OFFSETS INTO THE DISPATCH TABLE)
        ;
        .=0
000000  A.ACCE: .BLKW   1         ;I/O REQUEST ACCEPTANCE ENTRY POINT
000002  A.DEQU: .BLKW   1         ;I/O REQUEST DEQUEUE ENTRY POINT
000004  A.POWE: .BLKW   1         ;POWER FAILURE ENTRY POINT
000006  A.INPU: .BLKW   1         ;INPUT COMPLETION ENTRY POINT
000010  A.OUTP: .BLKW   1         ;OUTPUT COMPLETION ENTRY POINT
000012  A.CONN: .BLKW   1         ;CONNECTION ENTRY POINT
000014  A.DISC: .BLKW   1         ;DISCONNECTION ENTRY POINT
000016  A.RECE: .BLKW   1         ;INPUT CHARACTER RECEPTION ENTRY POINT
000020  A.PROC: .BLKW   1         ;INPUT CHARACTER PROCESSING ENTRY POINT
000022  A.CALL: .BLKW   1         ;CALL ACD BACK AFTER TRANSFER ENTRY POINT
```

PKTDF$ (Cont.)

```
;
; DEFINE THE STATUS BITS IN A.STA OF THE PROTOTYPE ACB
;
AS.DEL=1                        ;ACD IS MARKED FOR DELETE
AS.DIS=2                        ;ACD IS DISABLED

        .PSECT
```

SCBDF$

```
                SCBDF$  ,,SYSDEF

        ;
        ; STATUS CONTROL BLOCK
        ;
        ; THE STATUS CONTROL BLOCK (SCB) DEFINES THE STATUS OF A DEVICE
        ; CONTROLLER.  THERE IS ONE SCB FOR EACH CONTROLLER IN A SYSTEM.
        ; THE SCB IS POINTED TO BY UNIT CONTROL BLOCKS.  TO EXPAND ON THE
        ; TELETYPE EXAMPLE ABOVE, EACH TELETYPE INTERFACED VIA A DL11-A
        ; WOULD HAVE A SCB SINCE EACH DL11-A IS AN INDEPENDENT INTERFACE
        ; UNIT.  THE TELETYPES INTERFACED VIA THE DH11 WOULD ALSO EACH HAVE
        ; AN SCB SINCE THE DH11 IS A SINGLE CONTROLLER BUT MULTIPLEXES MANY
        ; UNITS IN PARALLEL.
        ;
                .IF NB  SYSDEF

                .ASECT
        .=0
000000  S.LHD:  .BLKW   2       ;CONTROLLER I/O QUEUE LISTHEAD
000004  S.URM:                  ;REFERENCE LABEL

                .IF DF  M$$PRO

                .BLKW   1       ;UNIBUS RUN MASK FOR THE FORK BLOCK

                .ENDC

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
000031  S.ROFF: .BLKB   1       ;OFFSET TO FIRST DEV REG TO COPY
000032  S.EMB:  .BLKW   1       ;ERROR MESSAGE BLOCK POINTER
000034  S.KTB:  .BLKW   1       ;START OF MULTI-ACCESS KRBS

                .PSECT

                .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 ENABLED (1=YES)
        S2.OPT=100              ;SUPPORTS SEEK OPTIMIZATION (1=YES)
        S2.CON=200              ;SCB AND KRB ARE CONTIGUOUS (1=YES)
```

SCBDF$ (Cont.)

```
        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


        ;
        ; 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 'B' SPINNING UP
        S3.OPT=200                ;SEEK OPTIMIZATION 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)


        ;
        ; MAPPING ASSIGNMENT BLOCK (FOR UNIBUS MAPPING REGISTER ASSIGNMENT)
        ;
                .ASECT
        .=0
000000  M.LNK:  .BLKW   1         ;LINK WORD
000002  M.UMRA: .BLKW   1         ;ADDRESS OF FIRST ASSIGNED UMR
000004  M.UMRN: .BLKW   1         ;NUMBER OF UMR'S ASSIGNED * 4
000006  M.UMVL: .BLKW   1         ;LOW 16 BITS MAPPED BY 1ST ASSIGNED UMR
000010  M.UMVH: .BLKB   1         ;HIGH 2 BITS MAPPED IN BITS 4 AND 5
000011  M.BFVH: .BLKB   1         ;HIGH 6 BITS OF PHYSICAL BUFFER ADDRESS
000012  M.BFVL: .BLKW   1         ;LOW 16 BITS OF PHYSICAL BUFFER ADDRESS
000014  M.LGTH=.                  ;LENGTH OF MAPPING ASSIGNMENT BLOCK

                .ENDC

                .PSECT
```

SHDDF$

```
                SHDDF$

        ;
        ; FIRST, WE MUST DEFINE THE I/O PACKET DEFINITIONS, SINCE WE
        ; USE THEM IN OUR DEFINITIONS.
        ;
                PKTDF$          ;DEFINE I/O PACKET DEFINITIONS


        ;
        ; SHADOW RECORDING LINKAGE BLOCK (UMB)
        ;
        ; THE UMB LINKS TOGETHER TWO UCB'S AS A SHADOW SET.  ONE IS THE
        ; PRIMARY UCB, THE OTHER THE SECONDARY UCB.  THE EXISTANCE OF A
        ; UMB SIGNALS THAT SHADOW RECORDING IS ENABLED ON A PARTICULAR
        ; UCB.
        ;
                .ASECT
        .=0
000000  M.LNK:  .BLKW   1       ;LINKAGE OF ALL UMB'S IN THE SYSTEM
000002  M.LHD:  .BLKW   1       ;LISTHEAD OF ALL ML NODES FOR THIS SET
000004  M.UCB:  .BLKW   2       ;PRIMARY AND SECONDARY UCB ADDRESSES
000010  M.STS:  .BLKW   1       ;STATUS WORD
000012  M.LBN:  .BLKB   1       ;HIGH ORDER BYTE OF FENCE
000013          .BLKB   1       ;UNUSED BYTE (MAYBE STATUS?)
000014          .BLKW   1       ;LOW ORDER WORD OF FENCE

000016  M.LGH=.


        ;
        ; UMB STATUS BIT DEFINITIONS
        ;
                .PSECT
        MS.MDA=1                ;UMB MARKED FOR DEALLOCATION (1=YES)
        MS.CHP=2                ;CATCHUP IN PROGRESS (1=YES)


        ;
        ; DEFINE THE OFFSETS FOR THE ML NODE, LINKED OFF OF THE UMB
        ; THROUGH CELL M.LHD.  THIS NODE CONTAINS THE SECONDARY I/O
        ; PACKET, AND DOUBLES AS THE ERROR PACKET TO THE ERROR MESSAGE
        ; TASK.
        ;
                .ASECT
        .=0
000000  ML.LNK: .BLKW   1       ;LINKAGE OF ALL ML NODES ON UMB
000002  ML.LEN: .BLKB   1       ;LENGTH OF ML NODE FOR DEALLOCATION
000003  ML.TYP: .BLKB   1       ;TYPE OF ML NODE FOR ERROR TASK
000004  ML.DNC: .BLKB   1       ;DONE COUNT OF PACKETS
000005          .BLKB   1       ;UNUSED
000006  ML.PRI: .BLKW   1       ;PRIMARY I/O PACKET ADDRESS
000010  ML.PKT: .BLKB   I.LGTH  ;SECONDARY I/O PACKET

000060  ML.LGH=.
```

SHDDF$ (Cont.)

```
;
; ML NODE TYPE CODES
;
        .PSECT
MT.PKT=1                ;ML NODE IS I/O PACKET TYPE


;
; I/O PACKET OFFSET DEFNS FOR USE BY SHADOW RECORDING
;
I.R0=I.PRM              ;STATUS STORAGE FOR R0 STATUS
I.R1=I.PRM+2            ;STATUS STORAGE FOR R1 STATUS


;
; DEFINE THE ERROR MESSAGE POINTERS THAT RESIDE IN THE I/O PACKET.
;
        .PSECT
ML.FID=ML.PKT+I.IOSB    ;FILE ID WHICH CONTAINS ERROR
ML.FSEQ=ML.PKT+I.IOSB+2 ;FILE SEQUENCE NUMBER OF FILE IN ERROR
ML.LBN=ML.PKT+I.PRM+10  ;HIGH ORDER LBN OF BLOCK(S) IN ERROR
ML.CNT=ML.PKT+I.PRM+4   ;NUMBER OF BLOCKS IN BAD XFER
ML.TCB=ML.PKT+I.TCB     ;TCB OF TASK WITH BAD REQUEST
ML.SR0=ML.PKT+I.R0      ;R0 OF SECONDARY I/O PACKET
ML.SR1=ML.PKT+I.R1      ;R1 OF SECONDARY I/O PACKET
ML.PR0=ML.PKT+I.PRM+14  ;R0 OF PRIMARY I/O PACKET
ML.PR1=ML.PKT+I.PRM+16  ;R1 OF PRIMARY I/O PACKET
```

TCBDF$

```
                TCBDF$  ,,SYSDEF

        ;
        ; TASK CONTROL BLOCK OFFSET AND STATUS DEFINITIONS
        ;
        ; TASK CONTROL BLOCK
        ;
                .ASECT
        .=0
000000  T.LNK:  .BLKW   1         ;UTILITY LINK WORD
000002  T.PRI:  .BLKB   1         ;TASK PRIORITY
000003  T.IOC:  .BLKB   1         ;I/O PENDING COUNT
000004  T.PCBV: .BLKW   1         ;POINTER TO COMMON PCB VECTOR
000006  T.NAM:  .BLKW   2         ;TASK NAME IN RAD50
000012  T.RCVL: .BLKW   2         ;RECEIVE QUEUE LISTHEAD
000016  T.ASTL: .BLKW   2         ;AST QUEUE LISTHEAD
000022  T.EFLG: .BLKW   2         ;TASK LOCAL EVENT FLAGS 1-32
000026  T.UCB:  .BLKW   1         ;UCB ADDRESS FOR PSEUDO DEVICE 'TI'
000030  T.TCBL: .BLKW   1         ;TASK LIST THREAD WORD
000032  T.STAT: .BLKW   1         ;FIRST STATUS WORD (BLOCKING BITS)
000034  T.ST2:  .BLKW   1         ;SECOND STATUS WORD (STATE BITS)
000036  T.ST3:  .BLKW   1         ;THIRD STATUS WORD (ATTRIBUTE BITS)
000040  T.DPRI: .BLKB   1         ;TASK'S DEFAULT PRIORITY
000041  T.LBN:  .BLKB   3         ;LBN OF TASK LOAD IMAGE
000044  T.LDV:  .BLKW   1         ;UCB ADDRESS OF LOAD DEVICE
000046  T.PCB:  .BLKW   1         ;PCB ADDRESS OF TASK PARTITION
000050  T.MXSZ: .BLKW   1         ;MAXIMUM SIZE OF TASK IMAGE (MAPPED ONLY)
000052  T.ACTL: .BLKW   1         ;ADDRESS OF NEXT TASK IN ACTIVE LIST
000054  T.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


        $$$=.                     ;MARK START OF PLAS AREA
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

                .IF NDF P$$LAS
        .=$$$                     ;MOVE LC BACK TO START OF PLAS AREA
                .ENDC


                .IF NB  SYSDEF


        $$$=.                     ;MARK START OF PARENT/OFFSPRING AREA
000104  T.OCBH: .BLKW   2         ;OFFSPRING CONTROL BLOCK LISTHEAD
000110  T.RDCT: .BLKW   1         ;OUTSTANDING OFFSPRING AND VT: COUNT

                .IF NDF P$$OFF
        .=$$$
                .ENDC
```

TCBDF$ (Cont.)

```
000112  T.SAST: .BLKW   1         ;SPECIFY AST LIST HEAD


        $$$=.
        T.RRM:  .BLKW   1         ;REQUIRED RUN MASK
        T.IRM:  .BLKW   1         ;INITIAL RUN MASK SET UP BY INSTALL
        T.CPU:  .BLKB   1         ;PROCESSOR NUMBER ON WHICH TASK LAST EXECUTD
                .BLKB   1         ;(UNUSED)

                .IF NDF M$$PRO
        .=$$$
                .ENDC


        $$$=.
        T.ACN:  .BLKW   1         ;POINTER TO ACCOUNTING BLOCK

                .IF NDF A$$CNT
        .=$$$
                .ENDC


        $$$=.
        T.ISIZ: .BLKW   1         ;SIZE OF ROOT I SPACE

                .IF NDF U$$DAS
        .=$$$
                .ENDC


        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


        ;
        ; TASK BLOCKING STATUS MASK
        ;
        TS.BLK=177777
```

TCBDF$ (Cont.)

```
;
; SECOND STATUS WORD (STATE BITS)
;
T2.AST=100000           ;AST IN PROGRESS (1=YES)
T2.DST=40000            ;AST RECOGNITION DISABLED (1=YES)
T2.CHK=20000            ;TASK NOT CHECKPOINTABLE (1=YES)
T2.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 FUNCT(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 SOFTWARE SERVICES USE
T3.GFL=1                ;GROUP GLOBAL EVENT FLAG LOCK


;
; STATUS BIT DEFINITIONS FOR FOURTH STATUS WORD (T.ST4)
;
T4.MUT=40               ;TASK IS A MULTI-USER TASK
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
```

TCBDF$ (Cont.)

```
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 L
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 B
TR.CPA=1                ;PROCESSOR A

        .ENDC

        .PSECT
```

UCBDF$

```
                UCBDF$  ,,TTDEF,SYSDEF

        ;
        ; UNIT CONTROL BLOCK
        ;
        ; THE UNIT CONTROL BLOCK (UCB) DEFINES THE STATUS OF AN INDIVIDUAL
        ; DEVICE UNIT AND IS THE CONTROL BLOCK THAT IS POINTED TO BY THE
        ; FIRST WORD OF AN ASSIGNED LUN.  THERE IS ONE UCB FOR EACH DEVICE
        ; UNIT OF EACH DCB.  THE UCB'S ASSOCIATED WITH A PARTICULAR DCB ARE
        ; CONTIGUOUS IN MEMORY AND ARE POINTED TO BY THE DCB.  UCB'S ARE
        ; VARIABLE LENGTH BETWEEN DCB'S BUT ARE OF THE SAME LENGTH FOR A
        ; SPECIFIC DCB.  TO FINISH THE TELETYPE EXAMPLE ABOVE, EACH UNIT ON
        ; BOTH INTERFACES WOULD HAVE A UCB.
        ;
                .ASECT
        .=177772

                .IF NB  SYSDEF

                .IF DF  A$$CNT

        .=177770
        U.UAB:  .BLKW   1       ;POINTER TO USER ACCOUNT BLOCK

                .IFF

        U.UAB:
                .ENDC

                .ENDC

177772  U.MUP:  .BLKW   1       ;MULTI-USER PROTECTION WORD
177774  U.LUIC: .BLKW   1       ;LOGIN UIC - MULTI USER SYSTEMS ONLY
177776  U.OWN:  .BLKW   1       ;OWNING TERMINAL - MULTI USER SYSTEMS ONLY
000000  U.DCB:  .BLKW   1       ;BACK POINTER TO DCB
000002  U.RED:  .BLKW   1       ;POINTER TO REDIRECT UNIT UCB
000004  U.CTL:  .BLKB   1       ;CONTROL PROCESSING FLAGS
000005  U.STS:  .BLKB   1       ;UNIT STATUS
000006  U.UNIT: .BLKB   1       ;PHYSICAL UNIT NUMBER
000007  U.ST2:  .BLKB   1       ;UNIT STATUS EXTENSION
000010  U.CW1:  .BLKW   1       ;FIRST DEVICE CHARACTERISTICS WORD
000012  U.CW2:  .BLKW   1       ;SECOND DEVICE CHARACTERISTICS WORD
000014  U.CW3:  .BLKW   1       ;THIRD DEVICE CHARACTERISTICS WORD
000016  U.CW4:  .BLKW   1       ;FOURTH DEVICE CHARACTERISTICS WORD
000020  U.SCB:  .BLKW   1       ;POINTER TO SCB
000022  U.ATT:  .BLKW   1       ;TCB ADDRESS OF ATTACHED TASK
000024  U.BUF:  .BLKW   1       ;RELOCATION BIAS OF CURRENT I/O REQUEST
000026          .BLKW   1       ;BUFFER ADDRESS OF CURRENT I/O REQUEST
000030  U.CNT:  .BLKW   1       ;BYTE COUNT OF CURRENT I/O REQUEST

000032  U.UCBX=U.CNT+2          ;POINTER TO UCB EXTENSION IN SECONDARY POOL
000034  U.ACP=U.CNT+4           ;ADDRESS OF TCB OF MOUNTED ACP
000036  U.VCB=U.CNT+6           ;ADDRESS OF VOLUME CONTROL BLOCK
000034  U.CBF=U.CNT+4           ;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
000054  U.BPKT=U.CNT+24         ;UNIT BAD BLOCK PACKET WAITING LIST
```

UCBDF$ (Cont.)

```
000060  U.UC2X=U.CNT+30          ;POINTER TO 2ND 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
        ; (ERROR LOGGING DEVICES ONLY)
        ;
        .=0
000000          .BLKW   9.       ;FIXED ACCOUNTING TRANSACTION HEADER
000022  X.NAME: .BLKW   2        ;DRIVE NAME IN RAD50
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


        ;
        ; 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

        X.DFFL=10.               ;DEFAULT FAIRNESS COUNT LIMIT
        X.DFSL=8.                ;DEFAULT SOFT ERROR LIMIT
        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  X.HSTI: .BLKW   2        ;HOST IDENTIFIER
000010  X.UNTI: .BLKW   4        ;UNIT IDENTIFIER
000020  X.MEDI: .BLKW   2        ;MEDIA IDENTIFIER
000024  X.SHUN: .BLKW   1        ;SHADOW UNIT
```

UCBDF$ (Cont.)

```
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   2       ;UNIT CYLINDER SIZE
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 EXTENTION


                .IF NB  TTDEF

        ;
        ; TERMINAL DRIVER DEFINITIONS
        ;
        .=U.BUF
000024  U.TUX:  .BLKW   1       ;POINTER TO UCB EXTENSION (UCBX)
000026  U.TSTA: .BLKW   3       ;STATUS TRIPLE-WORD
000034  U.TTAB: .BLKW   1       ;IF 0: U.TTAB+1 IS SINGLE-CHARACTER TYPE-
                                ;       AHEAD BUFFER, CURRENTLY EMPTY
                                ;IF ODD: U.TTAB+1 IS SINGLE-CHARACTER TYPE-
                                ;       AHEAD BUFFER AND HOLDS A CHARACTER
                                ;IF NON-0 AND EVEN: POINTER TO MULTI-
                                ;       CHARACTER TYPE-AHEAD BUFFER
        .=.-2                   ;THE NEXT TWO OFFSETS OVERLAP U.TTAB WHEN
                                ;THE TYPEAHEAD BUFFER IS IN SECONDARY POOL
000034  U.TECO: .BLKB   1       ;ECHO BUFFER FOR DMA OPERATIONS WHEN UCBX IS
                                ;IN SECONDARY POOL AND THUS NOT MAPPED BY A
                                ;UMR
000035  U.TBSZ: .BLKB   1       ;TYPEAHEAD BUFFER SIZE
000036  U.UIC:  .BLKW   1       ;DEFAULT UIC
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.ACB:  .BLKW   1       ;ANCILLARY CONTROL DRIVER BLOCK ADDR
000052  U.AFLG: .BLKW   1       ;ANCILLARY CONTROL DRIVER FLAGS WORD
000054  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
```

UCBDF$ (Cont.)

```
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              ;TYPE-AHEAD BUFFER ALLOCATION REQUESTED
S3.8BC=200              ;PASS 8 BITS ON INPUT
S3.RCU=400              ;RESTORE CURSOR (MUST = TF.RCU*400)
S3.ABD=1000             ;AUTO-BAUD SPEED DETECTION ENABLED
S3.ABP=2000             ;AUTO-BAUD SPEED DETECTION IN PROGRESS
S3.WAL=4000             ;WRITE-PASS-ALL (MUST = TF.WAL*400)
S3.VER=10000            ;LAST CHAR. IN TYPE-AHEAD BUFFER
                        ;HAS PARITY ERROR
S3.BCC=20000            ;LAST CHAR. IN TYPE-AHEAD BUFFER
                        ;HAS FRAMING ERROR
S3.DAO=40000            ;LAST CHAR. IN TYPE-AHEAD BUFFER
                        ;HAS DATA OVERRUN ERROR
                        ;NOTE - THE 3 BITS ABOVE MUST CORRESPOND
                        ;TO THE RESPECTIVE ERROR FLAGS IN THE
                        ;HARDWARE RECEIVE BUFFER
S3.PCU=100000           ;POSITION CURSOR BEFORE WRITE

        .ENDC
```

UCBDF$ (Cont.)

```
       ;
       ; 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


       ;
       ; 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               ;MASSBUS CONTROLLER (11M COMPATIBILITY ONLY)
       DV.EXT=400               ;UNIT ON EXTENDED 22-BIT UNIBUS CNTROLER
                                ;(1=YES)
       DV.SWL=1000              ;UNIT SOFTWARE WRITE LOCKED (1=YES)
       DV.ISP=2000              ;INPUT SPOOLED DEVICE (1=YES)
       DV.OSP=4000              ;OUTPUT SPOOLED DEVICE (1=YES)
       DV.PSE=10000             ;PSEUDO DEVICE (1=YES)
       DV.COM=20000             ;DEVICE IS MOUNTABLE AS COM CHANNEL (1=YES)
       DV.F11=40000             ;DEVICE IS MOUNTABLE AS F11 DEVICE (1=YES)
       DV.MNT=100000            ;DEVICE IS MOUNTABLE (1=YES)
```

UCBDF$ (Cont.)

```
;
; TERMINAL DEPENDENT CHARACTERISTICS WORD 2 (U.CW2) BIT DEFINITIONS
;
U2.DH1=100000           ;UNIT IS A MULTIPLEXER (1=YES)
U2.DJ1=40000            ;UNIT IS A DJ11 (1=YES)
U2.RMT=20000            ;UNIT IS REMOTE (1=YES)
U2.HFF=10000            ;UNIT HANDLES HARDWARE FORM FEEDS (1=YES)
U2.L8S=10000            ;OLD NAME FOR U2.HFF
U2.NEC=4000             ;DON'T ECHO SOLICITED INPUT (1=YES)
U2.CRT=2000             ;UNIT IS A CRT (1=YES)
U2.ESC=1000             ;UNIT GENERATES ESCAPE SEQUENCES (1=YES)
U2.LOG=400              ;USER LOGGED ON TERMINAL (0=YES)
U2.SLV=200              ;UNIT IS A SLAVE TERMINAL (1=YES)
U2.DZ1=100              ;UNIT IS A DZ11 (1=YES)
U2.HLD=40               ;TERMINAL IS IN HOLD SCREEN MODE (1=YES)
U2.AT.=20               ;MCR COMMAND AT. BEING PROCESSED (1=YES)
U2.PRV=10               ;UNIT IS A PRIVILEGED TERMINAL (1=YES)
U2.L3S=4                ;UNIT IS A LA30S TERMINAL (1=YES)
U2.VT5=2                ;UNIT IS A VT05B TERMINAL (1=YES)
U2.LWC=1                ;LOWER CASE TO UPPER CASE CONVERSION (0=YES)


;
; BIT DEFINITIONS FOR U.MUP
;
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 TYPE-
; AHEAD BUFFER
;
U.TAPR=24               ;OFFSET WITHIN UCB WHICH POINTS TO UCB EXT
U.TTBF=46               ;OFFSET WITHIN UCB EXTENSION WHICH POINTS TO
                        ;TYPEAHEAD BUFFER


;
; RH11-RS03/RS04 CHARACTERISTICS WORD 2 (U.CW2) BIT DEFINITIONS
;
U2.R04=100000           ;UNIT IS A RS04 (1=YES)


;
; RH11-TU16 CHARACTERISTICS WORD 2 (U.CW2) BIT DEFINITIONS
;
U2.7CH=10000            ;UNIT IS A 7 CHANNEL DRIVE (1=YES)


;
; TERMINAL DEPENDENT CHARACTERISTICS WORD 3 (U.CW3) BIT DEFINITIONS
;
U3.UPC=20000            ;UPCASE OUTPUT FLAG
```

UCBDF$ (Cont.)

```
;
; VIRTUAL TERMINAL 3RD CHARACTERISTICS WORD DEFINITIONS
;
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 DEFINITIONS
;
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)
US.FOR=40               ;UNIT IS MOUNTED AS FOREIGN VOLUME (1=YES)
US.MDM=20               ;UNIT IS MARKED FOR DISMOUNT (1=YES)


;
; CARD READER DEPENDENT UNIT STATUS BIT DEFINITIONS
;
US.ABO=1                ;UNIT IS MARKED FOR ABORT IF NOT READY
                        ;(1=YES)
US.MDE=2                ;UNIT IS IN 029 TRANSLATION NODE (1=YES)


;
; FILES-11 DEPENDENT UNIT STATUS BITS
;
US.WCK=10               ;WRITE CHECK ENABLED (1=YES)
US.SPU=2                ;UNIT IS SPINNING UP (1=YES)
US.VV=1                 ;VOLUME VALID IS SET (1=YES)


;
; 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)
```

UCBDF$ (Cont.)

```
;
; LPS11 DEPENDENT UNIT STATUS BIT DEFINITIONS
;
US.FRK=2                ;FORK IN PROGRESS (1=YES)
US.SHR=1                ;SHAREABLE FUNCTION IN PROGRESS (0=YES)


;
; ANSI MAGTAPE DEPENDANT UNIT STATUS BITS
;
US.LAB=4                ;UNIT HAS LABELED TAPE ON IT (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)


;
; 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
```

APPENDIX B

CONVERTING A USER-SUPPLIED RSX-11M DRIVER

This appendix describes the modifications that you must make to enable an RSX-11M user-supplied driver to run on an RSX-11M-PLUS system. The modifications involve both the driver data base and the driver code.

B.1 ASSUMPTIONS AND GENERAL APPROACH

The discussion in this appendix assumes that the RSX-11M user-supplied driver runs on a mapped system. Also, samples of code from the RL01/RL02 driver (DLDRV) are used as examples in this appendix.

As a general approach to converting a driver, proceed in the following manner:

1. Read the RSX-11M-PLUS Guide to Writing an I/O Driver to gain a feeling for the differences between RSX-11M and RSX-11M-PLUS drivers. Note especially the differences in the data structures (RSX-11M-PLUS has two additional structures).

2. Make the changes described in this appendix.

3. Incorporate the driver according to the guidelines given in Chapter 5.

For the purposes of this discussion, a standard disk driver is one that does not attempt to use any of the advanced driver features that are described in Chapter 1.

B.2 MODIFYING THE DATA BASE CODE

Before creating the driver data base, read the overview of programming user-written driver data bases (Section 4.2). It gives important information on ordering and labeling in the code.

B.2.1 Unit Control Block Changes

Ensure that the Unit Control Block (UCB) has the data needed for disk geometry calculations. Refer to the description of U.PRM in Section 4.4.4. The following is an example of the code needed to store the disk geometry:

```
        .BYTE   40.,2           ;U.PRM
        .WORD   512.            ;U.PRM+2
```

Notes:

1. 40. indicates the number of sectors per track.
2. 2 indicates the number of tracks per cylinder.
3. 512. indicates the number of cylinders per volume.
4. The values in the code are device dependent.

B.2.2 Status Control Block Changes

RSX-11M-PLUS requires a structure called the Controller Request Block (KRB). You can add the KRB data that RSX-11M-PLUS requires to the Status Control Block (SCB) data to effectively create one continuous structure. This arrangement is called the contiguous KRB/SCB and is described in Sections 4.2.4 and 4.4.7. Because the ordering of the SCB data differs from RSX-11M to RSX-11M-PLUS, you must rearrange the RSX-11M SCB data to accommodate the RSX-11M-PLUS requirements. If your driver refers to the SCB structures by symbolic offset and does not rely on physical ordering, you do not need to change the driver code that accesses the SCB. Refer to Sections 4.4.5 and 4.4.6 for a description of the offsets required.

There must be one KRB/SCB combination for each controller present on the system.

An example of code that includes the proper offsets appears in Figure B-1.

```
          .BYTE     PR5,160/4          ;K.PRI,K.VCT
          .BYTE     0*2,0              ;K.CON,K.IOC
          .WORD     KS.OFL             ;K.STS
$DLA::                                 ;START OF KRB
          .WORD     174400             ;K.CSR
          .WORD     DLA-$DLA           ;K.OFF
          .BYTE     0,0                ;K.HPU
          .WORD     .DL0               ;K.OWN
$DL0::                                 ;START OF CONTIGUOUS SCB
          .WORD     0,.-2              ;S.LHD/K.CRQ
          .WORD     0,0,0,0            ;S.FRK
          .WORD     0                  ;S.KS5
          .WORD     0                  ;S.PKT
          .BYTE     0,4.               ;S.CTM,S.ITM
          .BYTE     0,0                ;S.STS,S.ST3
          .WORD     S2.CON!S2.LOG      ;S.ST2
          .WORD     $DLA               ;S.KRB
          .BYTE     5.,0               ;S.RCNT,S.ROFF
          .WORD     0                  ;S.EMB
          .BLKW     6                  ;MAPPING ASSIGNMENT BLOCK
          .WORD     0                  ;KE.RHB
DLA:
```

Figure B-1 Contiguous KRB/SCB for DLDRV

Notes:

1. K.VCT and K.CSR can be changed dynamically by reconfiguration commands when you bring the device on-line. Refer to the *RSX-11M/M-PLUS System Management Guide*, Chapter 15 for information on the Reconfiguration task and commands.

2. Label DLA is used solely for calculating K.OFF (DLA-$DLA).

B.2.3 The Controller Table

RSX-11M-PLUS requires a structure called a Controller Table (CTB). Add the code to define the CTB according to the rules described in Sections 4.2.5 and 4.4.8. An example of the code needed to define the CTB appears in Figure B-2.

```
          .WORD     0                  ;L.ICB
DLCTB:                                 ;START OF CTB
          .WORD     0                  ;L.LNK
          .ASCII    /DL/               ;L.NAM
          .WORD     $DLDCB             ;L.DCB
          .BYTE     1,0                ;L.NUM,L.STS
$DLCTB::
          .WORD     $DLA               ;L.KRB
```

Figure B-2 Controller Table (CTB) for DLDRV

Notes:

1. The symbol $DLDCB is a pointer to the Device Control Block (DCB).

2. L.KRB points to the start of the KRB.

This example assumes that you have a loadable data base. If the data base is resident, you must include the CTB macro before L.LNK.

B.3 MODIFYING THE DRIVER CODE

Several changes must be made to the RSX-11M driver code. For an overview of the RSX-11M-PLUS coding requirements, refer to Section 4.3.

B.3.1 Conditional Symbols

You can remove most dependence on system conditional definitions from the code. RSX-11M-PLUS always defines the symbols D$$IAG, M$$MGE, M$$EXT, M$$MUP, and E$$DVC.

B.3.2 Controller-Dependent Code

At the I/O initiation entry point in RSX-11M drivers, you will find code for defining a table of UCB addresses and loading the UCB address of the currently active unit in the table. Remove this code and replace it with the GTPKT$ macro call. For guidelines on doing this, refer to Sections 4.3.2 and 4.5.2.

Following is an example of the RSX-11M driver code that you must remove:

```
        CALL    $GTPKT          ;GET AN I/O PACKET TO PROCESS
        BCC     1$              ;IF CC PROCESS REQUEST
        RETURN                  ;RETURN IF BUSY OR NO REQUEST
1$:     MOV     R5,CNTBL(R3)    ;SAVE ADDRESS OF REQUEST UCB
```

Insert the RSX-11M-PLUS GTPKT$ macro call, a sample of which follows:

```
        GTPKT$  DL,R$$L11       ;GET NEXT I/O PACKET TO PROCESS
```

B.3.3 Driver Dispatch Table

Replace the code that defines the entry point addresses with the DDT$ macro call. Refer to Section 4.3.1 for a description of the call and its parameters. Refer to Section 4.5.1 for a description of the Driver Dispatch Table (DDT) and the format of the labels that it uses for the entry points.

Following is an example of the RSX-11M driver code that you must replace:

```
$DLTBL::.WORD   DLINI           ;DEVICE INITIATOR ENTRY POINT
        .WORD   DLCAN           ;CANCEL I/O OPERATION ENTRY POINT
        .WORD   DLOUT           ;DEVICE TIMEOUT ENTRY POINT
        .WORD   DLPWF           ;POWERFAIL ENTRY POINT
```

Insert the RSX-11M-PLUS DDT$ macro call, an example of which follows:

```
        DDT$    DL,R$$L11       ;GENERATE DISPATCH TABLE
```

You do not have to add code to the driver to handle controller and unit status changes. The sample form of the macro call shown generates code to use the xxPWF entry point for controller and unit on-line and off-line status changes.

B.3.4 Reconfiguration Support

If you incorporate the device in the Reconfiguration task (HRC...) tables and the device calls the Executive volume valid routine, you must incorporate a local register pass routine in your driver, an example of which appears in Figure B-3.

```
;+
; MOVE THE CONTROLLER/DRIVE REGISTERS INTO THE
; SPECIFIED BUFFER.
;
; INPUTS:
;       R2 = CSR ADDRESS
;       R3 = BUFFER ADDRESS
;
; OUTPUTS:
;       R3 - ALTERED
;-

REGPAS: MOV     (R2),(R3)+        ;MOVE RLCS
        MOV     RLBA(R2),(R3)+    ;MOVE RLBA
        MOV     RLDA(R2),(R3)+    ;MOVE RLDA
        MOV     RLMP(R2),(R3)+    ;MOVE RLMP
        CALL    DLGST             ;EXECUTE GET DRIVE STATUS FUNCTION
        MOV     RLMP(R2),(R3)     ;SAVE DRIVE STATUS
        RETURN                    ;
```

Figure B-3 Register Pass Routine (REGPAS)

Notes:

1. The index values RLxxx and the subroutine DLGST are device specific.

B.3.5 Volume Valid Processing

If the device is a disk and has a volume valid function, the RSX-11M-PLUS Executive must be able to handle the correct function codes. Refer to the description of volume valid processing in Section 4.5.9. For volume valid support, you may also need to include the code that appears in Figure B-4.

```
        CALL    $VOLVD            ;VALIDATE VOLUME VALID
        BCS     IODON             ;IF CS WE FAILED
        TST     R0                ;TRANSFER FUNCTION?
        BMI     1$                ;IF MI YES
        TST     I.PRM+2(R1)       ;SIZE THE DISK
        BPL     IODON             ;IF PL NO
        MOV     S.CSR(R4),R2      ;RETRIEVE CSR ADDRESS
        CALL    DLRST             ;RESET DRIVE AND GET STATUS
        MOV     S.PKT(R4),R3      ;RETRIEVE I/O PACKET ADDRESS
        CALL    REGPAS            ;PASS REGISTERS TO HRC
        BR      IODON             ;FINISH I/O REQUEST
1$:                               ;REFERENCE LABEL
```

Figure B-4 Typical Handling of Volume Valid

Notes:

1. The subroutine DLRST is device specific.

B.3.6 Converting Logical Block Numbers

The $CVLBN routine converts a Logical Block Number (LBN) to a physical disk address. You can replace special-purpose code in the RSX-11M driver with a call to this Executive routine, a description of which is in Section 7.4.6. A sample of the code that you should remove appears in Figure B-5.

```
MOV     #40.,R1          ;DIVIDE BY SECTORS/SURFACE
CALL    $DIV             ;CALCULATE CYLINDER NUMBER
.REPT   6.
ASL     R0               ;POSITION CYLINDER AND SURFACE
.ENDR
BIS    R1,R0            ;MERGE SECTOR WITH CYLINDER AND SURFACE
```

Figure B-5 RSX-11M Logical Block Number Conversion

Figure B-6 includes the call to $CVLBN.

```
CALL    $CVLBN           ;CONVERT BLOCK NUMBER TO DISK ADDRESS
ROR     R1               ;PUT SURFACE BIT IN CARRY
ROL     R2               ;MERGE IT WITH THE CYLINDER NUMBER
ASH     #6,R2            ;POSITION CYLINDER AND SURFACE
BIS     R2,R0            ;MERGE SECTOR WITH CYLINDER AND SURFACE
```

Figure B-6 RSX-11M-PLUS Logical Block Number Conversion

B.3.7 Interrupt Entry Processing

Ensure that the INTSV$ macro call appears as the first line of code at each interrupt entry point in the driver. Refer to Section 4.3.3 for a description of the INTSV$ macro call and to Section 4.5.8 for a discussion of processing at an interrupt entry point. Following, is a sample INTSV$ macro call:

```
INTSV$  DL,PR5,R$$L11    ;;;SAVE REGISTERS AND SET PRIORITY
```

INDEX

RSX-11M-PLUS
Guide to Writing
an I/O Driver
AA-H267B-TC

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

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

Did you find errors in this manual? If so, specify the error and the page number.

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

- □ Assembly language programmer
- □ Higher-level language programmer
- □ Occasional programmer (experienced)
- □ User with little programming experience
- □ Student programmer
- □ Other (please specify) ______

Name ______ Date ______

Organization ______

Street ______

City ______ State ______ Zip Code ______
or Country

Do Not Tear - Fold Here and Tape

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

BSSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03061

Do Not Tear - Fold Here

Cut Along Dotted Line