RSX-11M
I/O Drivers Reference Manual

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

RSX-11M
I/O Drivers Reference Manual

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

RSX-11M Version 2

digital equipment corporation · maynard. massachusetts

First Printing, November 1974
Revised, September 1975

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

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

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

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

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM		

LIMITED RIGHTS LEGEND

Contract No. _____

Contractor or Subcontractor: Digital Equipment Corporation

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

CONTENTS

CONTENTS (Cont.)

CONTENTS (Cont.)

FIGURES

TABLES

PREFACE

0.1 MANUAL OBJECTIVES AND READER ASSUMPTIONS

This manual is designed to provide all information necessary to interface directly with the I/O device drivers supplied as part of the RSX-11M system. It is intended for use by experienced RSX-11M programmers who want to take advantage of the time and/or space savings which result from direct use of the I/O drivers.

The orientation of this manual is tutorial, but it does not attempt to introduce the reader to all areas of RSX-11M input/output operations. Readers are expected to be familiar with the RSX-11M Executive Reference Manual (DEC-11-OMERA-A-D) and to have some experience with the Task Builder and either FORTRAN IV or MACRO-11 assembly language. Readers should also be familiar with the PDP-11 terminology presented in the PDP-11 Processor Handbook and the PDP-11 Peripherals Handbook. Users of RSX-11M who do not require such detailed knowledge of the I/O drivers can use the device independent services provided by File Control Services (FCS) as documented in the RSX-11 I/O Operations Reference Manual (DEC-11-OMFSA-A-D).

Other manuals closely allied to the purposes of this document are described briefly in the RSX-11M/RSX-11S Documentation Directory, Order No. DEC-11-OMUGA-B-D. The Documentation Directory defines the intended readership of each manual in the RSX-11M/RSX-11S set and provides a brief synopsis of each manual's contents.

0.2 STRUCTURE OF THE DOCUMENT

This manual has three basic components:

1. Chapter 1 provides an overview of RSX-11M input/output operations. It introduces the reader to the use of logical unit numbers, directive parameter blocks, and macro calls. It describes all of the I/O functions common to a variety of devices, and summarizes standard error and status conditions relating to completion of I/O requests.

2. Chapters 2 through 14 describe the use of all device drivers supported by RSX-11M. These include the following:

Chapter	Device
2	Terminals and terminal communications line interfaces
3	Disks
4	DECtape
5	Magnetic tape
6	Cassette
7	Line printer
8	Card reader
9	Message-oriented communications line interfaces
10	Analog-to-digital converters
11	Universal digital controller
12	Laboratory peripheral systems
13	Paper tape reader/punch
14	Industrial control local and remote subsystems

Each of these chapters is structured in similar fashion and focuses on the following basic elements:

- Description of the device, including information on physical characteristics such as speed, capacity, access, and usage
- Summary of standard functions supported by the devices and descriptions of device-specific functions
- Discussion of special characters, carriage control codes, and functional characteristics, if relevant
- Summary of error and status conditions returned on acceptance or rejection of I/O requests
- Description of programming hints for users of the device under RSX-11M

3. Appendixes A through D provide quick reference material on I/O functions and status codes, a glossary of RSX-11M terms, and an example of RSX-11M I/O operations. These include the following:

Appendix	Contents
A	Summary of I/O functions by device
B	I/O function and status codes
C	Programming example
D	Glossary of RSX-11M terms

0.3 CONVENTIONS USED IN THIS MANUAL

There are a number of conventions and assumptions used in this manual to present syntax and program coding examples. These are described in the following list.

1. Brackets ([]) in syntactic models enclose optional parameters.

The following example illustrates this format:

```
ASTX$S  [err]
```

2. Braces ({ }) in syntactic models indicate that one of the items must be selected, as in the following:

$$\text{CALL}\begin{Bmatrix}\text{DOM}\\ \text{DOMW}\end{Bmatrix}\text{<inm,icont,idata,[idx],[isb],[lun]>}$$

3. An ellipsis (...) in a syntactic model or coding example indicates that parameters have been omitted. As used in this manual, an ellipsis in a QIO macro call indicates omission of standard QIO parameters described in section 1.4. This is illustrated below:

```
QIO$C  IO.RLV,...,<stadd,size>
```

4. Consecutive commas in a coding example indicate null arguments. The following illustrates this usage:

```
QIO$C  IO.ATT,6,,,,AST01
```

5. Commas indicating null trailing optional arguments may be omitted, as in the following:

```
QIO$C  IO.KIL,9.
```

6. Certain parameters are required but ignored by RSX-11M; this is necessary to maintain compatibility with RSX-11D. For example, in the following, the priority specification (fourth parameter) is ignored:

```
QIO$C  IO.WLB,8.,EV,,IOSB,ASTX,<IOBUF,NBUF>
```

7. With the exception of MACRO-11 coding examples, all numbers in the text of this manual are assumed to be decimal; octal radix is explicitly declared as in the following:

   An illegal logical block number has been specified for DECtape. The number exceeds 577 (1101 octal).

   In MACRO-11 coding examples, all numbers are assumed to be octal; decimal radix is explicitly designated by following the number with a decimal point, as in the following example:

```
QIO$C  IO.RDB,14.,,,IOSB,,<IOBUF,80.>
```

8. In FORTRAN subroutine models, parameters which begin with the letters i through n indicate integer variables, as in the following example:

```
CALL DRS (ibuf,ilen,imode,irate,iefn,imask,isb,
          [nbuf],istart],[istop])
```

In general, where both i and n prefixes are used in a call, the i form indicates the name of an array and the n form specifies the size of the array.

All integer arrays and variables are assumed to occupy one storage word per variable (i.e., INTEGER*2) and all real arrays and variables are assumed to occupy two storage words per variable (i.e., REAL*4).

CHAPTER 1

RSX-11M INPUT/OUTPUT

1.1 OVERVIEW OF RSX-11M I/O

The RSX-11M real-time Executive supports a wide variety of PDP-11 input and output devices, including disks, DECtapes, magnetic tapes, tape cassettes, line printers, card readers, and such laboratory and industrial devices as analog-to-digital converters, universal digital controllers, and laboratory peripheral systems. Drivers for these devices are supplied by Digital Equipment Corporation as part of the RSX-11M system software. This manual describes all of the device drivers supported by RSX-11M and the characteristics, functions, error conditions, and programming hints associated with each. PDP-11 devices not described in this manual can be added to basic RSX-11M configurations, but users must develop and maintain their own drivers for these devices.

Input/output operations under RSX-11M are extremely flexible and are as device- and function-independent as possible. Programs issue I/O requests to logical units which have been previously associated with particular physical device units. Each program or task is able to establish its own correspondence between physical device units and logical unit numbers (LUNs). I/O requests are queued as issued; they are subsequently processed according to the relative priority of the tasks which issued them. I/O requests can be issued from MACRO-11 or FORTRAN tasks by means of the File Control Services (for appropriate devices), or can be interfaced directly to an I/O driver by means of the QIO system directive.

All of the I/O services described in this manual are requested by the user in the form of QIO system directives. A function code included in the QIO directive indicates the particular input or output operation to be performed. I/O functions can be used to request such operations as:

- attaching or detaching a physical device unit for a task's exclusive use
- reading or writing a logical or virtual block of data
- canceling a task's I/O requests

A wide variety of device-specific input/output operations (e.g., reading DECtape in reverse, rewinding cassette tape) can also be specified via QIO directives.

1.2 PHYSICAL, LOGICAL, AND VIRTUAL I/O

There are three possible modes in which an I/O transfer can take place. These are physical, logical, and virtual.

Physical I/O concerns reading and writing data in the actual physical units accepted by the hardware (e.g., sectors on a disk). For most devices, physical I/O is identical to logical I/O. For example, the RK05 disk has sectors of 256 words, the same number as RSX-11M logical blocks for all disks. Thus, in this case, a logical block maps directly into a physical block. For other devices, the mapping is not one to one. The RF11 disk, for example, is word-addressable; however no physical I/O may be done with the RF11. Data is always written in 256-word logical blocks. Another example is the RX01 flexible disk. Data is recorded in physical sectors of 64 words each. Logical blocks are made up of four physical sectors.

Logical I/O concerns reading and writing data in blocks that are convenient for the operating system. In most cases, logical blocks map directly into physical blocks. For block-structured devices (e.g., disks), logical blocks are numbered beginning at zero (0). For nonblock-structured devices (e.g., terminals), logical blocks are not addressable.

Virtual I/O concerns reading and writing data to open files. In this case, the executive maps virtual blocks into logical blocks. For file-structured devices (disks or DECtapes) virtual blocks are logical blocks, but are numbered starting from one (1) and are relative to the file rather than to the device. For nonfile-structured devices, the mapping from virtual block to logical block is direct.

1.3 RSX-11M DEVICES

The devices listed below are supported by RSX-11M. Drivers are supplied for each of these devices, and I/O operations for them are described in detail in subsequent chapters of this manual.

1. A variety of terminals, including the following:

   - ASR-33 and ASR-35 Teletypes (1)
   - KSR-33 and KSR-35 Teletypes (1)
   - LA30 DECwriters (serial and parallel)
   - LA36 DECwriter
   - VT05B Alphanumeric Display Terminal
   - VT50 Alphanumeric Display Terminal
   - RT02 Data Entry Terminal
   - RT02-C Badge Reader and Data Entry Terminal

---

(1) Teletype is a registered trademark of the Teletype Corporation.

These terminals are supported on the following asynchronous line interfaces:

- DJ11 Asynchronous Communications Line Interface Multiplexer
- DH11 and DH11-DM11-BB Asynchronous Communications Line Interface Multiplexer
- DL11-A, DL11-B, DL11-C, and DL11-D Asynchronous Communications Line Interfaces

2. A variety of disks, including the following:
   - RF11/RS11 Fixed-Head Disk
   - RP11/RP02 Pack Disk
   - RP11/RP03 Pack Disk
   - RX11/RX01 Flexible Disk
   - RP04 Pack Disk
   - RS03 Fixed-Head Disk
   - RS04 Fixed-Head Disk
   - RK11/RK05 Cartridge Disk
3. TC11/TU56 DECtape
4. Two types of magnetic tape:
   - TU16 Magnetic Tape
   - TM11/TU10 Magnetic Tape
5. TA11 Tape Cassette
6. Three line printers:
   - LP11 Line Printer
   - LS11 Line Printer
   - LV11 Line Printer
7. CR11 Card Reader
8. Synchronous and asynchronous line interfaces:
   - DL11-E Asynchronous Communication Line Interface
   - DP11 Synchronous Communication Line Interface
   - DU11 Synchronous Communication Line Interface
9. Two analog-to-digital converters:
   - AFC11 Analog-to-Digital Converter
   - AD01-D Analog-to-Digital Converter

10. UDC11 Universal Digital Controller

11. Laboratory Peripheral Systems

    . AR11 Laboratory Peripheral System

    . LPS11 Laboratory Peripheral System

12. Paper Tape Devices

    . PC11 Paper Tape Reader/Punch

    . PR11 Paper Tape Reader

13. ICS/ICR Industrial Control Local and Remote Subsystems

1.4 LOGICAL UNITS

This section describes the construction of the logical unit table and the use of logical unit numbers.

1.4.1 Logical Unit Number

A logical unit number or LUN is a number which is associated with a physical device unit during RSX-11M I/O operations. For example, LUN 1 might be associated with one of the terminals in the system, LUNs 2, 3, 4, and 5 with DECtape drives, and LUNs 6, 7, and 8 with disk units. The association is a dynamic one; each task running in the system can establish its own correspondence between LUNs and physical device units, and can change any LUN-physical device unit association at any time. The flexibility of this association contributes heavily to RSX-11M device independence.

A logical unit number is simply a short name used to represent a logical unit-physical device unit association. Once the association has been made, the LUN provides a direct and efficient mapping to the physical device unit, and eliminates the necessity to search the device tables whenever the system encounters a reference to a physical device unit.

The user should remember that, although a LUN-physical device unit association can be changed at any time, reassignment of a LUN at run time causes pending I/O requests for the previous LUN assignment to be cancelled. It is the user's responsibility to verify that all outstanding I/O requests for a LUN have been serviced before that LUN is associated with another physical device unit.

1.4.2 Logical Unit Table

There is one logical unit table (LUT) for each task running in an RSX-11M system. This table is a variable-length block contained in the task header. Each LUT contains sufficient 2-word entries for the number of logical units specified by the user at task build time.

Each entry or slot contains a pointer to the physical device unit currently associated with that LUN. Whenever a user issues an I/O request, RSX-11M matches the appropriate physical device unit to the LUN specified in the call by indexing into the logical unit table by the number supplied as the LUN. Thus if the call specifies 6 as the LUN, RSX-11M accesses the sixth 2-word entry in the LUT and associates the I/O request with the physical device unit to which the entry points. The number of LUN assignments valid for a task ranges from zero to 255, but cannot be greater than the number of LUNs specified at task build time.

Figure 1-1 illustrates a typical logical unit table.

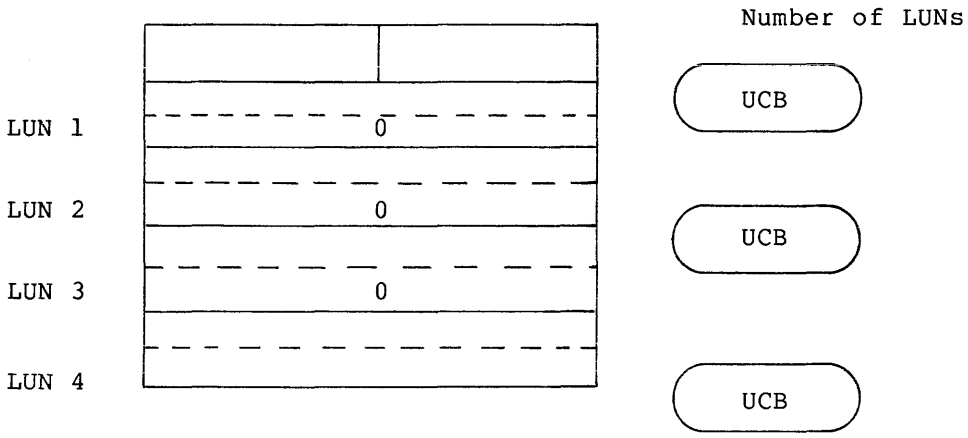


Figure 1-1
Logical Unit Table

Word 1 of each active (assigned) 2-word entry in the logical unit table points to the unit control block (UCB) of the physical device unit with which the LUN is associated. This linkage may be indirect - that is, the user may force redirection of references from one unit to another unit via the MCR command, REDIRECT. Word 2 of each entry is reserved for mountable devices.

1.4.3 Changing LUN Assignments

Logical unit numbers have no significance until they are associated with a physical device unit by means of one of the methods described below:

1. At task build time, the user can specify an ASG keyword option, which associates a physical device unit with a logical unit number referenced in the task being built.

2. The user or system operator can issue a REASSIGN command to MCR; this command reassigns a LUN to another physical device unit and thus changes the LUN-physical device unit correspondence. Note that this reassignment has no effect on the in-core image of a task.

3. At run time, a task can dynamically change a LUN assignment by issuing the ASSIGN LUN system directive, which changes the association of a LUN with a physical device unit during task execution.

1.5 ISSUING AN I/O REQUEST

User tasks perform I/O in the RSX-11M system by submitting requests for I/O service in the form of QIO system directives. See Chapter 2 of the RSX-11M Executive Reference Manual for a complete description of RSX-11M system directives.

In RSX-11M, as in most multiprogramming systems, tasks do not normally access physical device units directly. Instead, they utilize input/output services provided by the Executive, since it can effectively multiplex the use of physical device units over many users. The RSX-11M Executive routes I/O requests to the appropriate device driver and queues them according to the priority of the requesting task. I/O operations proceed concurrently with other activities in an RSX-11M system.

After an I/O request has been queued, the system does not wait for the operation to complete. If at any time the user task which issued the QIO request cannot proceed until the I/O operation has completed, it should specify an event flag (see sections 1.5.1 and 1.5.2) in the QIO request and should issue a WAITFOR system directive which specifies the same event flag at the point where synchronization must occur. The task then waits for completion of I/O by waiting for the specified event flag to be set.

Each QIO directive must supply sufficient information to identify and queue the I/O request. The user may also want to include parameters to receive error or status codes and to specify the address of an asynchronous system trap service routine. Certain types of I/O operations require the specification of device-dependent information as well. Typical QIO parameters are the following:

- I/O function to be performed
- Logical unit number associated with the physical device unit to be accessed
- Optional event flag number for synchronizing I/O completion processing
- Optional address of the I/O status block to which information indicating successful or unsuccessful completion is returned
- Optional address of an asynchronous system trap service routine to be entered on completion of the I/O request
- Optional device- and function-dependent parameters specifying such items as the starting address of a data buffer, the size of the buffer, and a block number

A set of system macros which facilitate the issuing of QIO directives is supplied with the RSX-11M system. These macros, which reside in the System Macro Library (SY:[1,1]RSXMAC.SML), must be made available to the source program by means of the MACRO-11 Assembler directive

.MCALL. The function of .MCALL is described in section 1.7.3. Several of the first six parameters in the QIO directive are optional, but space for these parameters must be reserved.

During expansion of a QIO macro, a value of zero is defaulted for all null (omitted) parameters. Inclusion of the device- and function-dependent parameters depends on the physical device unit and function specified. If the user wanted to specify only an I/O function code, a LUN, and an address for an asynchronous system trap service routine, the following might be issued:

```
QIO$C   IO.ATT,6,,,,ASTOX
```

where IO.ATT is the I/O function code for attach, 6 is the LUN, ASTOX is the AST address, and commas indicate null arguments for the event flag number, the request priority, and the address of the I/O status block. No additional device- or function-dependent parameters are required for an attach function. The C form of the QIO$ macro is used here and in most of the examples included in Chapter 1. Section 1.7 describes the three legal forms of the macro.

For convenience, any comma may be omitted if no parameters appear to the right of it. The command above could therefore be issued as follows, if the asynchronous system trap was not desired.

```
QIO$C   IO.ATT,6
```

All extra commas have been dropped. If, however, a parameter appears to the right of any place-holding comma, that comma must be retained.

1.5.1 QIO Macro Format

The arguments for a specific QIO macro call may be different for each I/O device accessed and for each I/O function requested. The general format of the call is, however, common to all devices and is as follows:

```
QIO$C   fnc,lun,[efn],[pri],[isb],[ast],[<p1,p2,...,p6>]
```

where brackets ([]) enclose optional or function-dependent parameters. If function-dependent parameters <p1,...,p6> are required, these parameters must be enclosed within angle brackets (<>). The following paragraphs summarize the use of each QIO parameter. Section 1.7 discusses different forms of the QIO$ macro itself.

The fnc parameter is a symbolic name representing the I/O function to be performed. This name is of the form:

```
IO.xxx
```

where xxx identifies the particular I/O operation. For example, a QIO request to attach the physical device unit associated with a LUN specifies the function code:

```
IO.ATT
```

A QIO request to cancel (or kill) all I/O requests for a specified LUN begins in the following way:

```
QIO$C   IO.KIL,...
```

The fnc parameter specified in the QIO request is stored internally as a function code in the high-order byte and modifier bits in the low-order byte of a single word. The function code is in the range zero through 31 and is a binary value supplied by the system to match the symbolic name specified in the QIO request. The correspondence between global symbolic names and function codes is defined in the system object module library. Local symbolic definitions may also be obtained via the FILIO$ and SPCIO$ macros which reside in the System Macro Library and are summarized in Appendix A. Several similar functions may have identical function codes, and may be distinguished only by their modifier bits. For example, the DECtape read logical forward and read logical reverse functions have the same function code. Only the modifier bits for these two operations are stored differently.

The lun parameter represents the logical unit number (LUN) of the associated physical device unit to be accessed by the I/O request. The association between the physical device unit and the LUN is specific to the task which issues the I/O request, and the LUN reference is usually device-independent. An attach request to the physical device unit associated with LUN 14 begins in the following way:

```
QIO$C  IO.ATT,14.,...
```

Because each task has its own logical unit table (LUT) in which the physical device unit-LUN correspondences are established, the legality of a lun parameter is specific to the task which includes this parameter in a QIO request. In general, the LUN must be in the following range:

$$0 \leq \text{LUN} \leq \text{length of task's LUT (if nonzero)}$$

The number of LUNs specified in the logical unit table of a particular task cannot exceed 255.

The efn parameter is a number representing the event flag to be associated with the I/O operation. It may optionally be included in a QIO request. The specified event flag is cleared when the I/O request is queued and is set when the I/O operation has completed. This allows the task to use the WAITFOR system directive to synchronize I/O programming by suspending execution to wait for an I/O operation to complete and efn to be set. However, if the task continues to execute, it may test the event flag whenever it chooses by using the READ ALL EVENT FLAGS system directive. If the user specifies an event flag number, this number must be in the range 1 through 64. If an event flag specification is not desired, efn can be omitted or can be supplied with a value of zero. Event flags 1 through 32 are local (specific to the issuing task); event flags 33 through 64 are global (shared by all tasks in the system). Flags 25 through 32 and 57 through 64 are reserved for use by system software. Within these bounds, the user can specify event flags as desired to synchronize I/O completion and task execution. Section 1.5.2 provides a more detailed explanation of event flags and significant events.

The optional pri parameter is supplied only to make RSX-11M QIO requests compatible with RSX-11D. A specific priority cannot be associated solely with the I/O request specified in the QIO macro call. An RSX-11M I/O request automatically assumes the priority of the requesting task. For consistency with RSX-11D, it is recommended that pri be valid, but the user should be aware that RSX-11M does not

use this specification in any way. RSX-11D priorities must be in range 1 through 250, and zero can be supplied to indicate the priority of the requesting task. A value of zero or a null specification is recommended for all RSX-11M use.

The optional isb parameter identifies the address of the I/O status block (I/O status double-word) associated with the I/O request. This block is a 2-word array in which a code representing the final status of the I/O request is returned on completion of the operation. This code is a binary value that corresponds to a symbolic name of the form IS.xxx (for successful returns) or IE.xxx (for error returns). The binary error code is returned to the low-order byte of the first word of the status block. It can be tested symbolically, by name. For example, the symbolic status IE.BAD is returned if a bad parameter is encountered. The following illustrates the examination of the I/O status block, IOST, to determine if a bad parameter has been detected.

```
QIO$C   IO.ATT,14.,2,,IOST,...
WTSE$C  2
        .
        .
        .
CMPB    #IE.BAD,IOST
BEQ     ERROR
```

The correspondence between global symbolic names and I/O completion codes is defined in the system object module library. Local symbolic definitions, which are summarized in Appendix B, may also be obtained via the IOERR$ macro which resides in the System Macro Library.

Certain device-dependent information is returned to the high-order byte of the first word of isb on completion of the I/O operation. If a read or write operation is successful, the second word is also significant. For example, in the case of a read function on a terminal, the number of bytes typed before a carriage return is returned in the second word of isb. If a Magtape unit is the device and a write function is specified, this number represents the number of bytes actually transferred. The status block can be omitted from a QIO request if the user does not intend to test for successful completion of the request.

The optional ast parameter specifies the address of a service routine to be entered when an asynchronous system trap occurs. Section 1.5.3 discusses the use of asynchronous system traps, and section 2.2.5 of the RSX-11M Executive Reference Manual describes traps in detail. If the user wants to interrupt his task to execute special code on completion of an I/O request, an asynchronous system trap routine can be specified in the QIO request. When the specified I/O operation completes, control branches to this routine at the software priority of the requesting task. The asynchronous code beginning at address ast is then executed, much as an interrupt service routine would be. If the user does not want to perform asynchronous processing, the ast parameter can be omitted or a value of zero specified in the QIO macro call.

The additional QIO parameters, <p1,p2,...,p6>, are dependent on the particular function and device specified in the I/O request. Between zero and six parameters can be included, depending on the particular I/O function. Rules for including these parameters and legal values are described in subsequent chapters of this manual.

1.5.2 Significant Events

"Significant event" is a term used in real-time systems to indicate a change in system status. In RSX-11M, a significant event is declared when an I/O operation completes. This signals the system that a change in status has occurred and indicates that the Executive should review the eligibility of all tasks in the system to determine which task should run next. The use of significant events helps cooperating tasks in a real-time system to communicate with each other and thus allows these tasks to control their own sequence of execution dynamically.

Significant events are normally set by system directives, either directly or indirectly, by completion of a specified function. Event flags associated with tasks may be used to indicate which significant event has occurred. Of the 64 event flags available in RSX-11M, the flags numbered 1 through 32 are local to an individual task and are set or reset only as a result of that task's operation. The event flags numbered 33 through 64 are common to all tasks. Flags 25 through 32 and 57 through 64 are reserved for RSX-11M system software use.

An example of the use of significant events follows. A task issues a QIO directive with an efn parameter specified. A WAITFOR directive follows the QIO and specifies as an argument the same event flag number. The event flag is cleared when the I/O request is queued by the Executive, and the task is suspended when it executes the WAITFOR directive until the event flag is set and a significant event is declared at the completion of the I/O request. The task resumes when the appropriate event flag is set, and execution resumes at the instruction following the WAITFOR directive. During the time that the task is suspended, tasks with priorities lower than that of the suspended task have a chance to run, thus increasing throughput in the system.

1.5.3 System Traps

System traps are used to interrupt task execution and to cause a transfer of control to another memory location for special processing. Traps are handled by the RSX-11M Executive and are relevant only to the task in which they occur. To use a system trap, a task must contain a trap service routine which is automatically entered when the trap occurs.

There are two types of system traps - synchronous and asynchronous. Both are used to handle error or event conditions, but the two traps differ in their relation to the task which is running when they are detected. Synchronous traps signal error conditions within the executing task. If the same instruction sequence were repeated, the same synchronous trap would occur. Asynchronous traps signal the completion of an external event such as an I/O operation. An asynchronous system trap (AST) usually occurs as the result of the initiation or completion of an external event rather than a program condition.

The Executive queues ASTs in a first-in-first-out queue for each task and monitors all asynchronous service routine operations. Because asynchronous traps are the end result of I/O-related activity, they cannot be controlled directly by the task which receives them.

However, the task may, under certain circumstances, block honoring an AST to prevent simultaneous access to a critical data region. When access to the critical data region has been completed, the queued ASTs may again be honored. The DSAR$S (DISABLE AST RECOGNITION) and ENAR$S (ENABLE AST RECOGNITION) system directives provide the mechanism for accomplishing this. An example of an asynchronous trap condition is the completion of an I/O request. The timing of such an operation clearly cannot be predicted by the requesting task. If an AST service routine is not specified in an I/O request, a trap does not occur and normal task execution continues.

Asynchronous system traps associated with I/O requests enable the requesting task to be truly event-driven. The AST service routine contained in the initiating task is executed as soon as possible, consistent with the system's priority structure. The use of the AST routine to service I/O related events provides a response time which is considerably better than a polling mechanism, and provides for better overlap processing than the simple QIO and WAITFOR sequence. Asynchronous system traps also provide an ideal mechanism for use in multiple buffering of I/O operations.

All ASTs are inserted in a first-in-first-out queue on a per task basis as they occur (i.e., the event which they are to signal has expired). They are effected one at a time whenever the task does not have ASTs disabled and is not already in the process of executing an AST service routine. The process of effecting an AST involves storing certain information on the task's stack, including the task's four WAITFOR mask words, the Directive Status Word (DSW), the PS, the PC and any trap dependent parameters. The task's general-purpose registers R0-R5 are not saved and thus it is the responsibility of the AST service routine to save and restore the registers it uses. After an AST is processed, the trap-dependent parameters (if any) must be removed from the task's stack and an AST SERVICE EXIT directive executed. The ASTX$S macro described in section 1.7.6 of this manual is used to issue the AST SERVICE EXIT directive. On AST service exit, control is returned to another queued AST, the executing task, or another task which has been waiting to run. Section 2.2.5 of the RSX-11M Executive Reference Manual describes in detail the purpose of AST service routines and all system directives used to handle them.

1.6 DIRECTIVE PARAMETER BLOCKS

A directive parameter block (DPB) is a fixed-length area of contiguous memory which contains the arguments specified in a system directive macro call. The DPB for a QIO directive has a length of 12 words. It is generated as the result of the expansion of a QIO macro call. The first byte of the DPB contains the directive identification code (DIC) - always 1 for QIO. The second byte contains the size of the directive parameter block in words - always 12 for QIO. During assembly of a user task containing QIO requests, the MACRO-11 Assembler generates a directive parameter block for each I/O request specified in a QIO macro call. At run time, the Executive uses the arguments stored in each DPB to create, for each request, an I/O packet in system dynamic storage. The packet is entered by priority into a queue of I/O requests for the specified physical device unit. This queue is created and maintained by the RSX-11M Executive and is ordered by the priority of the tasks which issued the requests. The I/O drivers examine their respective queues for the I/O request with the highest priority capable of being executed. This request is

de-queued (removed from the queue) and the I/O operation is performed. The process is then repeated until the queue is emptied of all requests.

After the I/O request has been completed, the Executive declares a significant event and may set an event flag, cause a branch to an asynchronous system trap service routine, and/or return the I/O status, depending on the arguments specified in the original QIO macro call. Figure 1-2 illustrates the layout of a sample DPB.

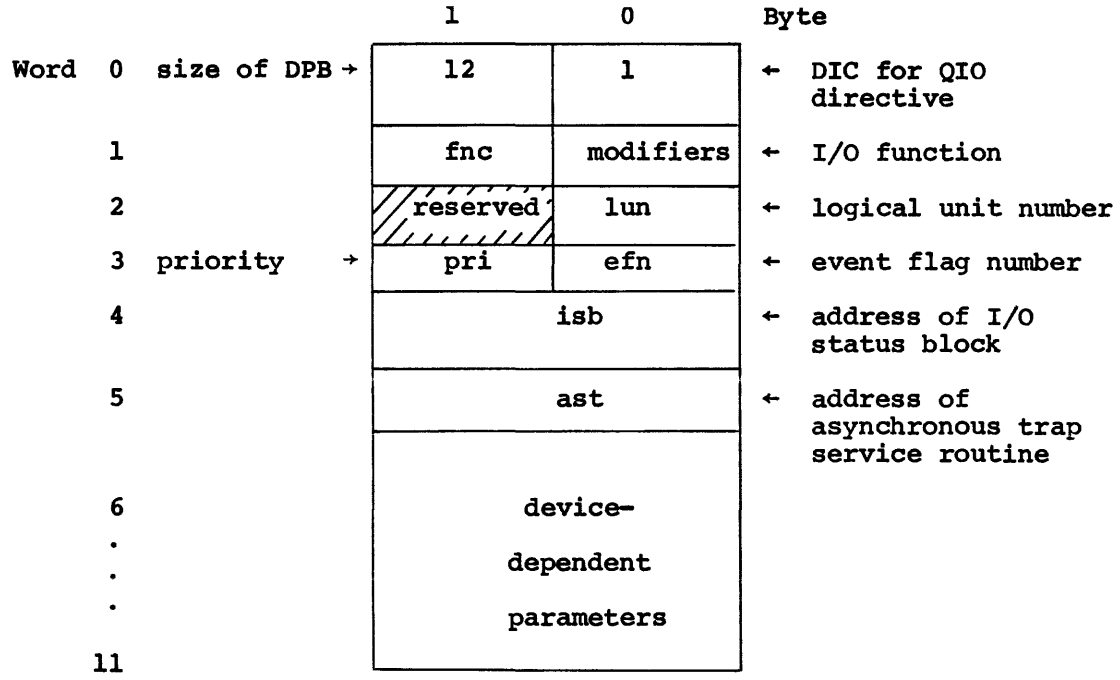


Figure 1-2
QIO Directive Parameter Block

1.7 I/O-RELATED MACROS

There are several system macros supplied with the RSX-11M system which are used to issue and return information about I/O requests. These macros reside in the System Macro Library and must be made available during assembly via the MACRO-11 assembler directive .MCALL.

There are three distinct forms of most of the system directive macros discussed in this section. The following list summarizes the forms of QIO$, but the characteristics of each form also apply to ALUN$, GLUN$, and other system directive macros described below.

1. QIO$ generates a directive parameter block for the I/O request at assembly time, but does not provide the instructions necessary to execute the request. This form of the request is actually executed using the DIR$ macro.

2. QIO$S generates a directive parameter block for the I/O request on the stack, and also generates code to execute the request. This is a useful form for reentrant, sharable code since the DPB is generated dynamically at execution time.

3. QIO$C generates a directive parameter block for the I/O request at assembly time, and also generates code to execute the request. The DPB is generated in a separate program section called $DPB$$. This approach incurs little system overhead and is useful when an I/O request is executed from only one place in the program.

Parameters for both the QIO$ and QIO$C forms of the macro must be valid expressions to be used in assembler data-generating directives such as .WORD and .BYTE. Parameters for the QIO$S form must be valid source operand address expressions to be used in assembler instructions such as MOV and MOVB. The following example references the same parameters in the three distinct forms of the macro call.

```
QIO$     IO.RLB,6,2,,,AST01,...

QIO$C    IO.RLB,6,2,,,AST01,...

QIO$S    #IO.RLB,#6,#2,,,#AST01,...
```

Only the QIO$S form of the macro produces the DPB dynamically. The other two forms generate the DPB at assembly time. The characteristics and use of these different forms are described in greater detail in the RSX-11M Executive Reference Manual.

The following Executive directives and assembler macros are described in this section:

1. QIO$, which is used to request an I/O operation and supply parameters for that request.

2. DIR$, which specifies the address of a directive parameter block as its argument, and generates code to execute the directive.

3. .MCALL, which is used to make available from the System Macro Library all macros referenced during task assembly.

4. ALUN$, which is used to associate a logical unit number with a physical device unit at run time.

5. GLUN$, which requests that the information about a physical device unit associated with a specified LUN be returned to a user-specified buffer.

6. ASTX$S, which is used to terminate execution of an asynchronous system trap (AST) service routine.

7. WTSE$, which instructs the system to suspend execution of the issuing task until a specified event flag is set.

1.7.1 The QIO$ Macro: Issuing an I/O Request

As described in section 1.7, there are three distinct forms of the QIO$ macro. QIO$S generates a DPB for the I/O request on the stack, and also generates code to execute the request. QIO$C generates a DPB and code, but the DPB is generated in a separate program section. QIO$ generates only the DPB for the I/O request. This form of the macro call is used in conjunction with DIR$ (see section 1.7.2) to

execute an I/O request. In the following example, the DIR$ macro actually generates the code to execute the QIO$ directive. It provides no QIO parameters of its own, but references the QIO directive parameter block at address QIOREF by supplying this label as an argument.

```
QIOREF:  QIO$    IO.RLB,6,2,,,AST01,...    ;  CREATE QIO DPB
         .
         .
         .
READ1:   DIR$    #QIOREF                   ;  ISSUE I/O REQUEST
         .
         .
         .
READ2:   DIR$    #QIOREF                   ;  ISSUE I/O REQUEST
```

1.7.2 The DIR$ Macro: Executing a Directive

The DIR$ (execute directive) macro has been implemented to allow a task to reference a previously defined directive parameter block without requiring that it specify all of the parameters of that macro again. It is issued in the form:

```
DIR$   [addr][,err]
```

where: addr is the address of a directive parameter block to be used in the directive. If addr is not included, the DPB itself or the address of the DPB is assumed to already be on the stack.

err is an optional argument which specifies the address of an error routine to which control branches if the directive is rejected. The branch occurs via a JSR PC, err.

1.7.3 The .MCALL Directive: Retrieving System Macros

.MCALL is a MACRO-11 assembler directive which is used to retrieve macros from the System Macro Library (SY:[1,1]RSXMAC.SML) for use during assembly. It must be included in every user task which invokes system macros. .MCALL is usually placed at the beginning of a user task source module and specifies, as arguments in the call, all system macros which must be made available from the library.

The following example illustrates the use of this directive:

```
         .MCALL  QIO$,QIO$S,DIR$,WTSE$S     ;  MAKE MACROS AVAILABLE
            .
            .
            .
QIOREF:  QIO$    IO.RLB,6,2,,,AST01, . . .  ;  CREATE ONLY QIO DPB
            .
            .
            .
```

```
READ1:  DIR$    #QIOREF                         ; ISSUE I/O REQUEST
          .
          .
          .
READ2:  QIO$S   #IO.ATT,#14.,#8.,,,#AST02       ; CREATE DPB ON STACK
                                                ; AND ISSUE REQUEST
          .
          .
          .
```

As many macro references as can fit on a line can be included in a single .MCALL directive. There is no limit to the number of .MCALL directives that can be specified.

1.7.4 The ALUN$ Macro: Assigning a LUN

The ASSIGN LUN macro is used to associate a logical unit number with a physical device unit at run time. All three forms of the macro call may be used. ASSIGN LUN does not request I/O for the physical device unit, nor does it attach the unit for exclusive use by the issuing task. It simply establishes a LUN-physical device unit relationship, so that when the task requests I/O for that particular LUN, the associated physical device unit is referenced. The macro is issued from a MACRO-11 program in the following way:

```
ALUN$   lun,dev,unt
```

where:

- lun is the logical unit number to be associated with the specified physical device unit.
- dev is the device name of the physical device or a logical device name assigned to a physical device (see MCR ASN command).
- unt is the unit number of that device specified above.

For example, to associate LUN 10 with terminal unit 2, the following macro call could be issued by the task:

```
ALUN$C  10.,TT,2
```

A unit number of 0 represents unit 0 for multi-unit devices such as disk, DECtape, or terminals; it indicates the single available unit for devices without multiple units, such as card readers and line printers.

The following list contains physical device names, listed alphabetically, that may be included as dev parameters for all standard devices supported by RSX-11M.

Name	Device
AD	AD01-D Analog-to-Digital Converter
AF	AFC11 Analog-to-Digital Converter
AR	AR11 Laboratory Peripheral System

CR	CR11 Card Reader
CT	TA11/TU60 Tape Cassette
DB	RP04 Pack Disk
DF	RF11/RS11 Fixed-Head Disk
DK	RK11/RK05 Cartridge Disk
DP	RP02/RP03 Pack Disk
DS	RS03 and RS04 Fixed-Head Disks
DT	TC11/TU56 DECtape
DX	RX11/RX01 Flexible Disk
IC	ICS/ICR Industrial Control Local and Remote Subsystems
LP	LP11, LS11, and LV11 Line Printers
LS	LPS11 Laboratory Peripheral System
MM	TU16 Magnetic Tape
MT	TM11/TU10 Magnetic Tape
PP	PC11 Paper Tape Punch
PR	PC11 or PR11 Paper Tape Reader
TT	Terminals
UD	UDC11 Universal Digital Controller
XB	DA11-B Parallel Unibus Link
XL	DL11-E Asynchronous Communication Line Interface
XP	DP11 Synchronous Communication Line Interface
XQ	DQ11 Synchronous Communication Line Interface
XU	DU11 Synchronous Communication Line Interface

A pseudo-device is a logical device which can normally be redirected by the operator to another physical device unit at any time, without requiring changes in programs which reference the pseudo-device. Dynamic redirection of a physical device unit affects all tasks in the system; reassignment by means of the MCR REASSIGN command affects only one task. The following pseudo-devices are supported by RSX-11M:

Code	Device
CL	Console listing, normally the line printer
CO	Console output, normally the main operator's console
TI	Pseudo-input terminal, normally the terminal from which a task was requested

SY	System default device, normally the disk from which the system was bootstrapped

The pseudo-device TI cannot be redirected, since such redirection would have to be handled on a per task rather than a system wide basis (i.e., change the TI device for one task without affecting the TI assignments for other tasks).

Logical devices are SYSGEN options of RSX-11M that allow the user to assign logical names to physical devices by means of the MCR command ASN. See the RSX-11M Operator's Procedures Manual for a full description.

The example included below illustrates the use of the three forms of the ALUN$ macro.

```
;
; DATA DEFINITIONS
;

ASSIGN: ALUN$   10.,TT,2        ; GENERATE DPB
        .
        .
        .

;
; EXECUTABLE SECTION
;

        DIR$    #ASSIGN         ; EXECUTE DIRECTIVE
        .
        .
        .
        ALUN$C  10.,TT,2        ; GENERATE DPB IN SEPARATE PROGRAM
        .                       ; SECTION, THEN GENERATE CODE TO
        .                       ; EXECUTE THE DIRECTIVE
        .
        ALUN$S  #10.,#"TT,#2    ; GENERATE DPB ON STACK, THEN
                                ; EXECUTE DIRECTIVE
```

1.7.5 The GLUN$ Macro: Retrieving LUN Information

The GET LUN INFORMATION macro requests that information about a LUN-physical device unit association be returned in a 6-word buffer specified by the issuing task. All three forms of the macro call may be used. It is issued from a MACRO-11 program in the following way:

```
GLUN$   lun,buf
```

where:

lun	is the logical unit number associated with the physical device unit for which information is requested.
buf	is the 6-word buffer to which information is returned.

For example, to request information on the disk unit associated with LUN 8, the following call is issued:

```
GLUN$C   8.,IOBUF
```

The 6-word buffer contains the following indicators on completion of the directive:

Word	Byte	Bit	Contents
0			Name of device associated with lun
1	0		Unit number of associated device
	1		Driver flag value, indicating that the driver is resident (always returned as 128 (200 octal) in RSX-11M)
2		0	Unit record-oriented device (e.g., card reader, line printer) (1 = yes)
		1	Carriage-control device (e.g., line printer, terminal) (1 = yes)
		2	Terminal device (1 = yes)
		3	Directory device (e.g., DECtape, disk) (1 = yes)
		4	Single directory device (1= yes)
		5	Sequential device (1 = yes)
		6-12	Reserved
		13	Device mountable as a communications channel for Digital network support (e.g., DP11, DU11) (1 = yes)
		14	Device mountable as a FILES-11 device (e.g., disk) (1 = yes)
		15	Device mountable (logical OR of bits 13 and 14) (1 = yes)
3			Undefined (included for RSX-11D compatibility)
4			Undefined (included for RSX-11D compatibility)
5			Default buffer size for device (e.g., length of line for terminal)

The example included below illustrates the use of the three forms of the GLUN$ macro.

```
;
; DATA DEFINITIONS
;

GETLUN: GLUN$   6,DSKBUF        ; GENERATE DPB
        .
        .
        .

;
; EXECUTABLE SECTION
;

        DIR$    #GETLUN         ; EXECUTE DIRECTIVE
        .
        .
        .
        GLUN$C  6,DSKBUF        ; GENERATE DPB IN SEPARATE PROGRAM
        .                       ; SECTION, THEN GENERATE CODE TO
        .                       ; EXECUTE THE DIRECTIVE
        .
        GLUN$S  #6,#DSKBUF      ; GENERATE DPB ON STACK, THEN
                                ; EXECUTE DIRECTIVE
```

1.7.6 The ASTX$S Macro: Terminating AST Service

The AST SERVICE EXIT macro is used to terminate execution of an asynchronous system trap (AST) service routine. Only the ASTX$S form of this macro is provided; ASTX$ and ASTX$C are unsupported forms of the macro call. The macro is issued in the following way:

```
ASTX$S  [err]
```

where: err is an optional argument which specifies the address of an error routine to which control branches if the directive is rejected.

On completion of the operation specified in this macro call, if another AST is queued and asynchronous system traps have not been disabled, then the next AST is immediately entered. Otherwise, the task's state before the AST was entered is restored (it is the AST service routine's responsibility to save and restore the registers it uses).

1.7.7 The WTSE$ Macro: Waiting for an Event Flag

The WAIT FOR SINGLE EVENT FLAG macro instructs the system to suspend execution of the issuing task until the event flag specified in the macro call is set. This macro is extremely useful in synchronizing activity on completion of an I/O operation. All three forms of the macro call may be used. It is issued as follows:

```
WTSE$   efn
```

where: efn is the event flag number

WTSE$ causes the task to be blocked from execution until the specified event flag is set. Frequently, an efn parameter is also included in a QIO$ macro call, and the event flag is set on completion of the I/O operation specified in that call. The following example illustrates task blocking until the setting of the specified event flag occurs. This example also illustrates the use of the three forms of the macro call.

```
;
; DATA DEFINITIONS
;

WAIT:   WTSE$   5               ; GENERATE DPB
IOSB:   .BLKW   2               ; I/O STATUS BLOCK
        .
        .
        .

;
; EXECUTABLE SECTION
;

        ALUN$S  #14.,#"MM       ; ASSIGN LUN 14 TO MAGTAPE UNIT ZERO
        QIO$C   IO.ATT,14.,5    ; ATTACH DEVICE
        DIR$    #WAIT           ; EXECUTE WAITFOR DIRECTIVE
        .
        .
        .
        QIO$S   #IO.RLB,#14.,#2,,#IOSB,#ASTX,<#BUF,#80.>
        .                       ; READ RECORD, USE EFN2
        .
        .
        WTSE$S  #2              ; WAIT FOR READ TO COMPLETE
        .
        .
        .
        QIO$C   IO.WLB,14.,3,,IOSB,AST01,<BUF,80.>
        .                       ; WRITE RECORD, USE EFN3
        .
        .
        WTSE$C  3               ; WAIT FOR WRITE TO COMPLETE
        QIO$C   IO.DET,14.      ; DETACH DEVICE
        .
        .
        .
```

1.8 STANDARD I/O FUNCTIONS

There are a large number of input/output operations that can be specified by means of the QIO directive. A particular operation can be requested by including the appropriate function code as the first parameter of a QIO macro call. Certain functions are standard. These functions are almost totally device-independent and can thus be requested for nearly every device described in this manual. Others are device-dependent and are specific to the operation of only one or two I/O devices. This section summarizes the function codes and characteristics of the following device-independent I/O operations:

- attach to an I/O device
- detach from an I/O device
- cancel I/O requests
- read a logical block
- read a virtual block
- write a logical block
- write a virtual block

For certain physical device units discussed in this manual, a standard I/O function may be described as being a NOP. This means that no operation is performed as a result of specifying the function, and an I/O status code of IS.SUC is returned in the I/O status block specified in the QIO macro call.

In the following descriptions and in formats shown in subsequent chapters, the five parameters represented by the ellipsis (...) are as explained in section 1.5.1.

1.8.1 IO.ATT: Attaching to an I/O Device

The function code IO.ATT is specified by a user task when that task requires exclusive use of an I/O device. This function code is included as the first parameter of a QIO macro call in the following way:

```
QIO$C   IO.ATT,...
```

Successful completion of an IO.ATT request causes the specified physical device unit to be dedicated for exclusive use by the issuing task. This enables the task to process input or output in an unbroken stream and is especially useful on sequential, nonfile-oriented devices such as terminals, card readers, and line printers. An attached physical device unit remains under control of the issuing task until it is explicitly detached by that task. The same LUN must be specified for both the attach and detach functions.

While a physical device unit is attached, the I/O driver for that unit dequeues only I/O requests issued by the task that issued the attach. Thus, a request to attach a device unit already attached by another task will not be processed until the attachment is broken and no higher priority request exists for the unit. A LUN that is associated with an attached physical device unit may not be reassigned by means of an ASSIGN LUN directive.

If the task which issued an attach function exits or is aborted before it issues a corresponding detach, the RSX-11M Executive automatically detaches the physical device unit.

1.8.2 IO.DET: Detaching from an I/O Device

The function code IO.DET is used to detach a physical device unit which has been previously attached by means of an IO.ATT request for exclusive use of the issuing task. This function code is included as the first parameter of a QIO macro call in the following way:

```
        QIO$C   IO.DET,...
```

The LUN specifications of both IO.ATT and IO.DET must be the same, as in the following example, which also illustrates the use of "S" forms of several macro calls.

```
        .MCALL  ALUN$S,QIO$S
        ALUN$S  #14.,#"CR       ; ASSOCIATE CARD READER WITH LUN 14
        .
        .
        .
        QIO$S   #IO.ATT,#14.    ; ATTACH CARD READER
        .
        .
        .
LOOP:   QIO$S   #IO.RLB,#14.,... ; READ CARD
        .
        .
        .
        QIO$S   #IO.DET,#14.    ; DETACH CARD READER
```

1.8.3 IO.KIL: Canceling I/O Requests

The function code IO.KIL is issued by a task to cancel all of that task's I/O requests for a particular physical device unit, including all pending and active requests. This results in the status code IE.ABO being returned in the I/O status block and the event flag being set (if specified) for the respective requests, but does not initiate any asynchronous system trap (AST) service routine that may have been specified. Whether the current request is actually cancelled depends on the device. Because file-structured devices (disk and DECtape) operate quickly, IO.KIL is a NOP for these devices and simply causes the return of IS.SUC in the I/O status block.

This function code is included as the first parameter of a QIO macro in the following way:

```
        QIO$C   IO.KIL,...
```

IO.KIL is useful in such special cases as canceling an I/O request on a physical device unit from which a response is overdue (i.e., a read on a terminal).

1.8.4 IO.RLB: Reading a Logical Block

The function code IO.RLB is specified by a task to read a block of data from the physical device unit specified in the macro call. This function code is included as the first parameter of a QIO macro in the following way:

```
        QIO$C   IO.RLB,...,<stadd,size,pn>
```

where: stadd is the starting address of the data buffer.

size is the data buffer size in bytes.

pn represents one to four optional parameters, used to specify such additional information as block numbers for certain devices.

1.8.5 IO.RVB: Reading a Virtual Block

The function code IO.RVB is used to read a virtual block of data from the physical device unit specified in the macro call. A "virtual" block indicates a relative block position within a file and is identical to a "logical" block for such sequential devices as terminals and card readers. It is recommended that all tasks use virtual rather than logical reads. However, if a virtual read is issued for a file-structured device (disk or DECtape), the user must ensure that a file is open on the specified physical device unit. This function code is included as the first parameter of a QIO macro call in the following way:

```
QIO$C   IO.RVB,...,<stadd,size,pn>
```

where: stadd is the starting address of the data buffer.

size is the data buffer size in bytes.

pn represents one to four optional parameters, used to specify such additional information as block numbers for certain devices.

1.8.6 IO.WLB: Writing a Logical Block

The function code IO.WLB is specified by a task to write a block of data to the physical device unit specified in the macro call. This function code is included as the first parameter of a QIO macro call in the following way:

```
QIO$C   IO.WLB,...,<stadd,size,pn>
```

where: stadd is the starting address of the data buffer.

size is the data buffer in bytes.

pn represents one to four optional parameters, used to specify such additional information as block numbers or format control characters for certain devices.

1.8.7 IO.WVB: Writing a Virtual Block

The function code IO.WVB is used to write a virtual block of data to a physical device unit. A "virtual" block indicates a relative block position within a file and is identical to a "logical" block for such sequential devices as terminals and line printers. It is recommended that all tasks use virtual rather than logical writes. However, if a

virtual write is issued for a file-structured device (disk or DECtape), the user must ensure that a file is open on the specified physical device unit. This function code is included as the first parameter of a QIO macro call in the following way:

```
QIO$C   IO.WVB,...,<stadd,size,pn>
```

where: stadd is the starting address of the data buffer.

size is the data buffer size in bytes.

pn represents one to four optional parameters, used to specify such additional information as block numbers or format control characters for certain devices.

1.9 I/O COMPLETION

When an I/O request has been completed, either successfully or unsuccessfully, one or more actions may be taken by the Executive. Selection of return conditions depends on the parameters included in the QIO macro call. There are three major returns:

1. A significant event is declared on completion of an I/O operation. If an efn parameter was included in the I/O request, the corresponding event flag is set.

2. If an isb parameter was specified in the QIO macro call, a code identifying the type of success or failure is returned in the low-order byte of the first word of the I/O status block at the location represented by isb.

   This status return code is of the form IS.xxx (success) or IE.xxx (error). For example, if the device accessed by the I/O request is not ready, a status code of IE.DNR is returned in isb. The section below (Return Codes) summarizes general codes returned by most of the drivers described in this manual.

   If the isb parameter was omitted, the requesting task cannot determine whether the I/O request was successfully completed. A carry clear return from the directive itself simply means that the directive was accepted and the I/O request was queued, not that the actual input/output operation was successfully performed.

3. If an ast parameter was specified in the QIO macro call, a branch to the asynchronous system trap (AST) service routine which begins at the location identified by ast occurs on completion of the I/O operation. See section 1.5.3 for a detailed description of AST service routines.

1.10 RETURN CODES

There are two kinds of status conditions recognized and handled by RSX-11M when they occur in I/O requests:

- Directive conditions, which indicate the acceptance or rejection of the QIO directive itself
- I/O status conditions, which indicate the success or failure of the I/O operation

Directive conditions relevant to I/O operations may indicate any of the following:

- directive acceptance
- invalid buffer specification
- invalid efn parameter
- invalid lun parameter
- invalid DIC number or DPB size
- unassigned LUN
- insufficient memory

A code indicating the acceptance or rejection of a directive is returned to the directive status word at symbolic location $DSW. This location can be tested to determine the type of directive condition.

I/O conditions indicate the success or failure of the I/O operation specified in the QIO directive. I/O driver errors include such conditions as device not ready, privilege violation, file already open, or write-locked device. If an isb parameter is included in the QIO directive, identifying the address of a 2-word I/O status block, an I/O status code is returned in the low-order byte of the first word of this block on completion of the I/O operation. This code is a binary value which corresponds to a symbolic name of the form IS.xxx or IE.xxx. The low-order byte of the word can be tested symbolically, by name, to determine the type of status return. The correspondence between global symbolic names and directive and I/O completion status codes is defined in the system object module library. Local symbolic definitions may also be obtained via the DRERR$ and IOERR$ macros which reside in the System Macro Library and are summarized in Appendix B.

Binary values of status codes always have the following meaning:

Code	Meaning
Positive (greater than zero)	Successful completion
Zero	Operation still pending
Negative	Unsuccessful completion

A pending operation means that the I/O request is still in the queue of requests for the respective driver, or the driver has not yet completely serviced the request.

1.10.1 Directive Conditions

Table 1-1 summarizes the directive conditions which may be encountered in QIO directives. The acceptance condition is first, followed by error codes indicating various reasons for rejection, in alphabetical order.

Table 1-1
Directive Conditions

Code	Reason
IS.SUC	Directive accepted The first six parameters of the QIO directive were valid, and sufficient dynamic memory was available to allocate an I/O packet. The directive is accepted.
IE.ADP	Invalid address The I/O status block or the QIO DPB was outside of the issuing task's address space or was not aligned on a word boundary.
IE.IEF	Invalid event flag number The efn specification in a QIO directive was less than zero or greater than 64.
IE.ILU	Invalid logical unit number The lun specification in a QIO directive was invalid for the issuing task. For example, there were only five logical unit numbers associated with the task, and the value specified for lun was greater than five.
IE.SDP	Invalid DIC number or DPB size The directive identification code (DIC) or the size of the directive parameter block (DPB) was incorrect; the legal range for a DIC is from 1 through 127, and all DIC values must be odd. Each individual directive requires a DPB of a certain size. If the size is not correct for the particular directive, this code is returned. The size of the QIO DPB is always 12 words.
IE.ULN	Unassigned LUN The logical unit number in the QIO directive was not associated with a physical device unit. The user may recover from this error by issuing a valid ASSIGN LUN directive and then reissuing the rejected directive.

Table 1-1 (Cont.)
Directive Conditions

Code	Reason
IE.UPN	Insufficient dynamic memory There was not enough dynamic memory to allocate an I/O packet for the I/O request. The user can try again later by blocking the task with a WAITFOR SIGNIFICANT EVENT directive. Note that WAITFOR SIGNIFICANT EVENT is the only effective way for the issuing task to block its execution, since other directives that could be used for this purpose themselves require dynamic memory for their execution (e.g., MARK TIME).

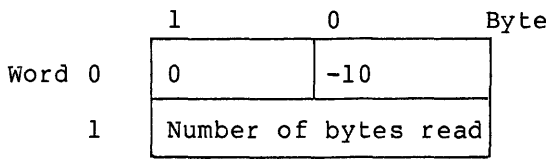
1.10.2 I/O Status Conditions

The following list summarizes status codes which may be returned in the I/O status block specified in the QIO directive on completion of the I/O request. The I/O status block is a 2-word block with the following format:

- The low-order byte of the first word receives a status code of the form IS.xxx or IE.xxx on completion of the I/O operation.
- The high-order byte of the first word is usually device-dependent; in cases where the user might find information in this byte helpful, this manual identifies that information.
- The second word contains the number of bytes transferred or processed if the operation is successful and involves reading or writing.

If the isb parameter of the QIO directive is omitted, this information is not returned.

The following illustrates a sample 2-word I/O status block on completion of a terminal read operation:



where -10 is the status code for IE.EOF (end of file). If this code is returned, it indicates that input was terminated by typing CTRL/Z, which is the end-of-file termination sequence on a terminal.

To test for a particular error condition, the user generally compares the low-order byte of the first word of the I/O status block with a symbolic value as in the following:

```
CMPB    #IE.DNR,IOSB
```

However, to test for certain types of successful completion of the I/O operation, the entire word value must be compared. For example, if a carriage return terminated a line of input from the terminal, a successful completion code of IS.CR is returned in the I/O status block. If an ESCape or ALTMODE character was the terminator, a code of IS.ESC is returned. To check for either of these codes, the user should first test the low-order byte of the first word of the block for IS.SUC and then test the full word for IS.CR or IS.ESC.

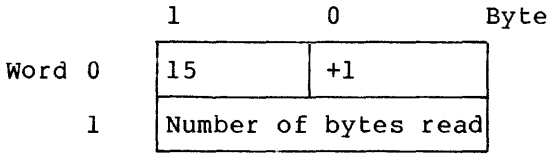
Note that all three of the following comparisons will test equal since the low-order byte in all cases is +1.

```
CMP     #IS.CR,IOSB

CMP     #IS.ESC,IOSB

CMPB     #IS.SUC,IOSB
```

In the case of a successful completion where the carriage return is the terminal indicator (IS.CR), the following illustrates the status block:



where 15 is the octal code for carriage return and +1 is the status code for successful completion.

The codes described in Table 1-2 are general status codes which apply to the majority of devices presented in subsequent chapters. Error codes specific to only one or two drivers are described only in relation to the devices for which they are returned. The list below describes successful and pending codes first, then error codes in alphabetical order.

Table 1-2
I/O Status Conditions

Code	Reason
IS.SUC	Successful completion The I/O operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading or writing.
IS.PND	I/O request pending The I/O operation specified in the QIO directive has not yet been executed. The I/O status block is filled with zeros.

Table 1-2 (Cont.)
I/O Status Conditions

Code	Reason
IE.ABO	Operation aborted The specified I/O operation was cancelled via IO.KIL while in progress or while still in the I/O queue.
IE.ALN	File already open The task attempted to open a file on the physical device unit associated with the specified LUN, but a file has already been opened by the issuing task on that LUN.
IE.BAD	Bad parameter An illegal specification was supplied for one or more of the device-dependent QIO parameters (words 6-11). For example, a bad channel number or gain code was specified in an analog-to-digital converter I/O operation.
IE.BLK	Illegal block number An illegal block number was specified for a file-structured physical device unit. This code is returned, for example, if block 4800 is specified for an RK05 disk, on which legal block numbers extend from zero through 4799.
IE.BYT	Byte-aligned buffer specified. Byte alignment was specified for a buffer, but only word alignment is legal for the physical device unit. For example, a disk function requiring word alignment was requested, but the buffer was aligned on a byte boundary. Alternately, the length of a buffer was not an appropriate multiple of bytes. For example, all RP03 disk transfers must be an even multiple of four bytes.
IE.DAA	Device already attached The physical device unit specified in an IO.ATT function was already attached to the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task.
IE.DNA	Device not attached The physical device unit specified in an IO.DET function was not attached to the issuing task. This code has no bearing on the attachment status with respect to other tasks.

Table 1-2 (Cont.)
I/O Status Conditions

Code	Reason
IE.DNR	Device not ready The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation. This code is often returned as the result of an interrupt timeout, that is, a "reasonable" amount of time has passed, and the physical device unit has not responded.
IE.EOF	End-of-file encountered An end-of-file mark, record, or control character was recognized on the input device.
IE.IFC	Illegal function A function code was specified in an I/O request that was illegal for the specified physical device unit. This code is returned if the task attempts to execute an illegal function or if, for example, a read function is requested on an output-only device, such as the line printer.
IE.NLN	File not open The task attempted to close a file on the physical device unit associated with the specified LUN, but no file was currently open on that LUN.
IE.NOD	Insufficient buffer space Dynamic storage space has been depleted, and there was insufficient buffer space available to allocate a secondary control block. For example, if a task attempts to open a file, buffer space for the window and file control block must be supplied by the Executive. This code is returned when there is not enough space for such an operation.
IE.OFL	Device off-line The physical device unit associated with the LUN specified in the QIO directive was not on-line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.OVR	Illegal read overlay request A read overlay was requested and the physical device unit specified in the QIO directive was not the physical device unit from which the task was installed. The read overlay function can only be executed on the physical device unit from which the task image containing the overlays was installed.

Table 1-2 (Cont.)
I/O Status Conditions

Code	Reason
IE.PRI	Privilege violation The task which issued a request was not privileged to execute that request. For example, for the UDC11 and LPS11, a checkpointable task attempted to connect to interrupts or to execute a synchronous sampling function.
IE.SPC	Illegal address space The buffer requested for a read or write request was partially or totally outside the address space of the issuing task. Alternately a byte count of zero was specified.
IE.VER	Unrecoverable error After the system's standard number of retries have been attempted upon encountering an error, the operation still could not be completed. This code is returned in the case of parity, CRC, or similar errors.
IE.WLK	Write-locked device The task attempted to write on a write-locked physical device unit.

CHAPTER 2

TERMINAL DRIVER

2.1 INTRODUCTION

The terminal driver provides support for a variety of terminal devices under RSX-11M. Table 2-1 summarizes the terminals supported, and subsequent sections describe these devices in greater detail.

Table 2-1
Standard Terminal Devices

Model	Column Width	Character Set	Baud Range
ASR-33/35	72	64	110
KSR-33/35	72	64	110
LA30-P	80	64	300
LA30-S	80	64	110-300
LA36	80-132	64-96*	110-300
RT02	64	64	110-1200
RT02-C	64	64	110-1200
VT05B	72	64*	110-2400
VT50	72	64	110-9600

Where appropriate, terminals must be set to transmit only upper-case alphabetic characters. Input lines can be at most 80 bytes; longer input lines are truncated. The terminal driver supports the communication line interfaces summarized in Table 2-2. These interfaces are described in greater detail in section 2.7. Programming is identical for all.

---

* Both upper and lower case input are supported.

Table 2-2
Standard Communication Line Interfaces

Model	Type
DH11	16-line multiplexer or
DH11-DM11-BB	16-line multiplexer with modem control
DJ11	16-line multiplexer
DL11-A/B/C/D	Single-line interfaces

2.1.1 ASR-33/35 Teletypes

The ASR-33 and ASR-35 Teletypes are asynchronous hard-copy terminals. No paper tape reader or punch capability is supported.

2.1.2 KSR-33/35 Teletypes

The KSR-33 and KSR-35 Teletypes are asynchronous, hard-copy terminals.

2.1.3 LA30 DECwriters

The LA30 DECwriter is an asynchronous, hard-copy terminal that is capable of producing an original and one copy. It is particularly appropriate for systems requiring large numbers of printer-terminals. The LA30-P is a parallel model and the LA30-S is a serial model.

2.1.4 LA36 DECwriter

The LA36 DECwriter is an asynchronous terminal which produces hard copy and operates in serial mode. It has an impact printer capable of generating multipart and special preprinted forms. Both upper-case and lower-case characters can be received and transmitted.

2.1.5 RT02 Alphanumeric Display Terminal and RT02-C Badge Reader/ Alphanumeric Display Terminal

The RT02 is a compact, alphanumeric display terminal designed for applications in which source data is primarily numeric. A shift key permits the entry of 30 discrete characters, including upper-case alphabetic characters. The RT02 can, however, receive and display 64 characters.

The RT02-C model also contains a badge reader. This option provides a reliable method of identifying and controlling access to the PDP-11 or to a secure facility. Furthermore, data in a format corresponding to that of a badge (22-column fixed data) can be entered very quickly.

2.1.6 VT05B Alphanumeric Display Terminal

The VT05B is an alphanumeric display terminal that consists of a CRT display and a self-contained keyboard. From a programming point of view, it is equivalent to other terminals, except that the VT05B offers direct cursor addressing.

2.1.7 VT50 Alphanumeric Display Terminal

The VT50 is an alphanumeric display terminal that consists of a CRT display and a keyboard. It is similar to the VT05B in operation, but does not offer direct cursor addressing.

2.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the GET LUN INFORMATION system directive (the first characteristics word) contains the following information for terminals. A setting of 1 indicates that the described characteristic is true for terminals.

Bit	Setting	Meaning
0	1	Record-oriented device
1	1	Carriage-control device
2	1	Terminal device
3	0	Directory device
4	0	Single-directory device
5	0	Sequential device
6-12	0	Reserved
13	0	Device mountable as a communications channel
14	0	Device mountable as a FILES-11 volume
15	0	Device mountable

Words 3 and 4 are undefined; word 5 indicates the default buffer size for the device, for terminals the width of the terminal carriage or display screen.

2.3 QIO MACRO

Table 2-3 lists the standard and device-specific functions of the QIO macro that are valid for terminals.

Table 2-3
Standard and Device-Specific QIO Functions for Terminals

STANDARD FUNCTIONS:

Format	Function
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O requests
QIO$C IO.RLB,...,<stadd,size>	Read logical block (Read typed input into buffer)
QIO$C IO.RVB,...,<stadd,size>	Read virtual block (Read typed input into buffer)
QIO$C IO.WLB,...,<stadd,size,vfc>	Write logical block (Print buffer contents)
QIO$C IO.WVB,...,<stadd,size,vfc>	Write virtual block (Print buffer contents)

DEVICE-SPECIFIC FUNCTIONS:

QIO$C IO.RAL,...,<stadd,size>	Read logical block. Pass all bits. Do not intercept control characters or mask out parity bit.
QIO$C IO.WAL,...,<stadd,size>	Write logical block. Pass all bits. Do not intercept control characters or add parity bit.

where: stadd is the starting address of the data buffer (may be on a byte boundary).

size is the data buffer size in bytes (must be greater than zero).

vfc is a vertical format control character from Table 2-7.

The effect of IO.KIL on an in-progress request depends upon whether the request is for input or output. If it is for input (i.e., IO.RLB, IO.RVB, or IO.RAL), the request is forced to terminate, IE.ABO is returned, and the second word of the I/O status block contains the number of bytes already typed. If the request is for output (i.e., IO.WLB, IO.WVB, or IO.WAL), the transfer is terminated, and IS.SUC is returned.

2.4 STATUS RETURNS

Table 2-4 lists error and status conditions that are returned by the terminal driver described in this chapter.

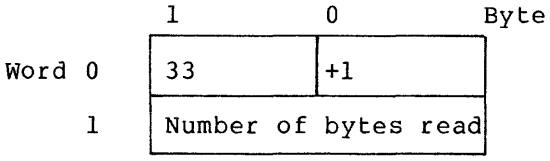
Table 2-4
Terminal Status Returns

Code	Reason
IE.EOF	Successful completion on a read with End-of-file The line of input read from the terminal was terminated with the end-of-file character CTRL/Z.
IS.SUC	Successful completion The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading or writing.
IS.CR	Successful completion on a read The line of input read from the terminal was terminated by a carriage return.
IS.ESC	Successful completion on a read The line of input read from the terminal was terminated by an ESCape or ALTMODE character.
IS.PND	I/O request pending The operation specified in the QIO directive has not yet been executed. The I/O status block if filled with zeros.
IE.ABO	Operation aborted The specified I/O operation was cancelled via IO.KIL while in progress or while in the I/O queue.
IE.DAA	Device already attached. The physical device unit specified in an IO.ATT function was already attached by the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task.
IE.DNA	Device not attached The physical device unit specified in an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.

Table 2-4 (Cont.)
Terminal Status Returns

Code	Reason
IE.DNR	Device not ready The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation. This code is returned to indicate one of the following conditions: . A timeout occurred on the physical device unit (i.e., an interrupt was lost). . An attempt was made to perform a transfer on a remote DH11 line without carrier present.
IE.IFC	Illegal function A function code was specified in an I/O request that was illegal for terminals.
IE.NOD	Buffer allocation failure Dynamic storage has been depleted, and there was insufficient space available to allocate a buffer for an input request (i.e., all input is buffered in the terminal driver).
IE.OFL	Device off-line The physical device unit associated with the LUN specified in the QIO directive was not on-line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.SPC	Illegal address space The buffer specified for a read or write request was partially or totally outside the address space of the issuing task. Alternately, a byte count of zero was specified.

The following illustrates the contents of the I/O status block on return of an IS.ESC code:



where 33 is the octal representation of the ESCape or ALTMODE character, and +1 is the status code for successful completion (IS.SUC). The contents of this block on return of IS.CR are the same, except that the high-order byte of word 0 contains 15, the octal code for carriage return. Unlike other RSX-11M return codes, IS.CR and IS.ESC are word values, rather than byte values. The low-order byte

simply indicates successful completion, and the high-order byte is required to show the specific type. To test for an IS.ESC or IS.CR code, the user can first test the low-order byte of the first word of the I/O status block for IS.SUC, and then test the full word for IS.ESC or IS.CR.

2.5 CONTROL CHARACTERS AND SPECIAL KEYS

This section describes the particular meaning of special terminal control characters and keys for RSX-11M. Note that control characters and special keys are not recognized by the driver during a read/pass all request (IO.RAL).

2.5.1 Control Characters

A control character is input from a terminal by holding the control key (CTRL) down while typing one other key. Two of the control characters described in Table 2-5, CTRL/U and CTRL/Z, are echoed on the terminal printer as ^U and ^Z respectively. Other control characters are recognized by the terminal driver but are not printing characters and are therefore not echoed.

Table 2-5
Terminal Control Characters

Character	Meaning
CTRL/C	Typing CTRL/C on the terminal causes unsolicited input on that terminal to be directed to the Monitor Console Routine (MCR). When the unsolicited input completes, it is passed to the MCR dispatcher. "MCR>" is echoed when the terminal is ready to accept the unsolicited input.
CTRL/I	Typing CTRL/I initiates a horizontal tab, and the terminal spaces to the next tab stop. Tabs are set at every eighth character position.
CTRL/J	Typing CTRL/J is equivalent to typing the LINE FEED key on the terminal.
CTRL/K	Typing CTRL/K initiates a vertical tab, and the terminal performs four line feeds.
CTRL/L	Typing CTRL/L initiates a form feed, and the terminal performs eight line feeds. Paging is not performed.
CTRL/M	Typing CTRL/M is equivalent to typing the carriage RETURN key on the terminal (See section 2.5.2).
CTRL/O	Typing CTRL/O suppresses output being sent to a terminal by the current I/O request. For attached terminals, CTRL/O remains in effect, and output continues to be suppressed until any of the following occur:

Table 2-5 (Cont.)
Terminal Control Characters

Character	Meaning
	. The terminal is detached . Solicited input is entered . Unsolicited input is entered . Another CTRL/O character is typed For unattached terminals, CTRL/O suppresses output for only the current output buffer.
CTRL/Q	Typing CTRL/Q resumes terminal output previously suspended by means of CTRL/S.
CTRL/S	Typing CTRL/S causes terminal output to be suspended. Output is resumed by typing CTRL/Q.
CTRL/U	Typing CTRL/U before typing a line terminator causes previously typed characters to be deleted back to the beginning of the line. The system echoes this character as ^U, followed by a carriage return and a line feed. This allows the line to be retyped.
CTRL/Z	Typing CTRL/Z indicates an end-of-file for the current terminal input. It signals MAC, PIP, TKB, and other system tasks that terminal input is complete and the task should exit. The system echoes this character as ^Z followed by a carriage return and a line feed.

2.5.2 Special Keys

The ESCape, carriage RETURN, and RUBOUT keys have special significance for terminal input, as described in Table 2-6. A line can be terminated by an ESCape (or ALTMODE) character, by a carriage RETURN, by CTRL/Z, or by completely filling the input buffer (i.e., exhausting the byte count before a line terminator is typed). The standard buffer size for a terminal can be determined by issuing a GET LUN INFORMATION system directive and examining word 5 of the information buffer.

Table 2-6
Special Terminal Keys

Key	Meaning
ESC	Typing ESCape or ALTMODE signals the terminal driver that there is no further input on the current line. This line terminator allows further input on the same line since the carriage or cursor is not returned to the first column position.
RETURN	Typing RETURN terminates the current line and causes the carriage or cursor to return to the first column on the line.

Table 2-6 (Cont.)
Special Terminal Keys

Key	Meaning
RUBOUT	Typing RUBOUT deletes the last character typed on an input line. Only characters typed since the last line terminator may be deleted. Several characters can be deleted in sequence by typing successive RUBOUTs. The first RUBOUT echoes as a backslash (\), followed by the character which has been deleted. Subsequent RUBOUTs cause only the deleted character to be echoed. The next character typed which is not a RUBOUT causes another \ followed by the new character to be echoed. The following example illustrates rubbing out ABC and then typing CBA: ABC\CBA\CBA The second backslash is not displayed if a line terminator is typed after rubbing out the characters on a line, as in the following: ABC\CBA

2.6 VERTICAL FORMAT CONTROL

Table 2-7 below summarizes the meanings of all characters used for vertical format control on the terminal. Any one of these characters can be specified as the value of the vfc parameter in an IO.WLB or IO.WVB function.

Table 2-7
Vertical Format Control Characters

Octal Value	Character	Meaning
040	blank	SINGLE SPACE - Output a line feed, print the contents of the buffer, and output a carriage return. Normally, printing immediately follows the previously printed line.
060	zero	DOUBLE SPACE - Output two line feeds, print the contents of the buffer, and output a carriage return. Normally, the buffer contents are printed two lines below the previously printed line.
061	one	PAGE EJECT - Output eight line feeds, print the contents of the buffer, and output a carriage return.
053	plus	OVERPRINT - Print the contents of the buffer and output a carriage return, normally overprinting the previous line.

Table 2-7 (Cont.)
Vertical Format Control Characters

Octal Value	Character	Meaning
044	dollar sign	PROMPTING OUTPUT - Output a line feed and print the contents of the buffer. This mode of output is intended for use with a terminal where a prompting message is output and input is then read on the same line.
000	null	INTERNAL VERTICAL FORMAT - The buffer contents are printed without addition of vertical format control characters. In this mode, more than one line of guaranteed contiguous output can be printed per I/O request.

All other vertical format control characters are interpreted as blanks (octal 040).

2.7 TERMINAL INTERFACES

This section summarizes the characteristics of the three types of standard communication line interfaces supported by RSX-11M.

2.7.1 DH11 Asynchronous Serial Line Multiplexer

The DH11 multiplexer interfaces up to 16 asynchronous serial communications lines for terminal use. The DH11 supports programmable baud rates with no parity. The DM11-BB option may be included to provide modem control for dial-up lines. These lines must be interfaced via Bell 103 or equivalent modems.

2.7.2 DJ11 Asynchronous Serial Line Multiplexer

The DJ11 multiplexer interfaces as many as 16 asynchronous serial lines to the PDP-11 for local terminal communications. The DJ11 does not provide a dial-up capability but supports jumper selectable baud rates.

2.7.3 DL11 Asynchronous Serial Line Interface

The DL11 supports a single asynchronous serial line and handles full-duplex communication between the PDP-11 and a terminal. There are 13 standard baud rates available to DL11 users (40-9600 baud). Four versions of the DL11 interface are supported by RSX-11M for terminal use: DL11-A, DL11-B, DL11-C, and DL11-D. The DL11-E is supported only for message-oriented communications and is described in Chapter 9.

2.8 PROGRAMMING HINTS

This section contains information on important considerations relevant to users of the terminal driver described in this chapter.

2.8.1 Terminal Line Truncation

If the number of characters to be printed exceeds the line length of the physical device unit, the terminal driver discards the excess characters until it receives one that instructs it to return to horizontal position 1. The user can determine that this will happen by examining word 5 of the information buffer returned by the GET LUN INFORMATION system directive.

2.8.2 ESCape Code Conversion

An ESCape or ALTMODE character code of 33, 175, or 176 is converted internally to 33 before it is returned to the user on input.

2.8.3 RT02-C Control Function

When sending a control character (e.g., vertical tab) to the RT02-C Badge Reader and Data Entry Terminal, the high-order bit (bit 7) of the byte must be set to one. This causes the terminal driver not to recognize the character. In the case of a vertical tab, 213 octal must be output rather than 13 octal.

CHAPTER 3

DISK DRIVERS

3.1 INTRODUCTION

the RSX-11M disk drivers support the disks summarized in Table 3-1. Subsequent sections describe these devices in greater detail.

Table 3-1
Standard Disk Devices

MODEL	RPM	SURFACES	CYLINDERS	WORDS/ TRACK	WORDS/ DRIVE
RF11/RS11	1800	1	128	2048.	262,144.
RP04	3600	19	411	5632.	43,980,288.
RS03	3600	1	64	4096.	262,144.
RS04	3600	1	64	8192.	524,288.
RK11/RK05	1500	2	200	3072.	1,228,800.
RP11E/RP02	2400	20	200	2560.	10,240,000.
RP11C/RP03	2400	20	400	2560.	20,480,000.
RX11/RX01	360	1	77	1664.	128,128.

All of the disks described in this chapter are accessed in essentially the same manner. Up to eight disks of each type (except RX01) may be connected to their respective controllers. Disks and other file-structured media under RSX-11M are divided logically into a series of 256-word blocks.

3.1.1 RF11/RS11 Fixed-Head Disk

The RF11 controller/RS11 fixed-head disk provides random-access bulk storage. It features fast track-switching time and a redundant set of timing tracks. The RF11/RS11 is unique because the hardware can automatically perform a spiral read across disk platters.

3.1.2 RP04 Pack Disk

The RP04 (RH11-RH70 controller/RP04 pack disk) pack disk consists of 19 data surfaces and a moving read/write head. It is similar to the RP11-C/RP03, but has twice the capacity. The RP04 offers large capacity storage with rapid access time.

3.1.3 RS03 Fixed-Head Disk

The RS03 (RH11-RH70 controller/RS03 fixed-head disk) is a fixed head disk which offers speed and efficiency. With 64 tracks per platter, and recording on one surface, the RS03 has a capacity of 262,144 words.

3.1.4 RS04 Fixed-Head Disk

The RS04 (RH11-RH70 controller/RS04 fixed-head disk) is similar to the RS03 disk, and interfaces to the same controller; but provides twice the number of words per track by recording on both surfaces of the platter, and thus twice the capacity.

3.1.5 RK11/RK05 Cartridge Disk

The RK11 controller/RK05 DECpack cartridge disk is an economical storage system for medium-volume, random-access storage. The removable disk cartridge offers the flexibility of large off-line capacity with rapid transfers of files between on- and off-line units without necessitating copying operations.

3.1.6 RP11/RP03 or RP02 Pack Disk

The RP11 controller/RP02 or RP03 pack disk consists of 20 data surfaces and a moving read/write head. The RP03 has twice as many cylinders, and thus, double the capacity of the RP02. Only an even number of words can be transferred in a read/write operation.

3.1.7 RX11/RX01 Flexible Disk

The RX11 controller/RX01 flexible disk is an economical storage system for low-volume, random-access storage. Data is stored in 26 64-word sectors per track; there are 77 tracks per disk. Data may be accessed by physical sector or logical block. If logical or virtual block I/O is selected, the driver reads four physical sectors. These sectors are interleaved to optimize data transfer. The next logical sector that falls on a new track is skewed by six sectors to allow for track to track switch time. Physical block I/O provides no interleaving or skewing and provides access to all 2002 sectors on the disk. Logical or virtual I/O starts on track one and provides access to 494 logical blocks.

3.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the GET LUN INFORMATION system directive (the first characteristics word) contains the following information for disks. A bit setting of 1 indicates that the described characteristic is true for disks.

Bit	Setting	Meaning
0	0	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	1	Directory device
4	0	Single-directory device
5	0	Sequential device
6-12	0	Peserved
13	0	Device mountable as a communications channel
14	1	Device mountable as a FILES-11 volume
15	1	Device mountable

Words 3 and 4 of the buffer are undefined; word 5 indicates the default buffer size, which is 512 for all disks.

3.3 QIO MACRO

This section summarizes the standard, and device-specific QIO functions for disk drivers.

3.3.1 Standard QIO Functions

Table 3-2 lists the standard functions of the QIO macro that are valid for disks.

Table 3-2
Standard QIO Functions for Disks

Format	Function
QIO$C IO.ATT,...	Attach device*
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Not applicable (NOP)
QIO$C IO.RLB,...,<stadd,size,,blkh,blkl>	Read logical block
QIO$C IO.RVB,...,<stadd,size,,blkh.blkl>	Read virtual block
QIO$C IO.WLB,...,<stadd,size,,blkh,blkl>	Write logical block
QIO$C IO.WVB,...,<stadd,size,,blkh,blkl>	Write virtual block

where: stadd is the starting address of the data buffer (must be on a word boundary).

size is the data buffer size in bytes (must be even, greater than zero, and, for the RP02 and RP03, also a multiple of four bytes).

blkh/blkl are block high and block low, combining to form a double-precision number that indicates the logical/virtual block address on the disk where the transfer starts; blkh represents the high eight bits of the address, and blkl the low 16 bits.

IO.RVB and IO.WVB are associated with file operations (see the RSX-11 I/O Operations Reference Manual). For these functions to be executed, a file must be open on the specified LUN if the volume associated with the LUN is mounted. Otherwise, the virtual I/O request is converted to a logical I/O request using the specified block numbers.

3.3.2 Device-Specific QIO Functions

The device-specific functions of the QIO macro are valid for the RX11 only; they are shown in Table 3-3.

*Only unmounted volumes may be attached. An attempt to attach a mounted volume will result in an IE.PRI status being returned in the I/O status doubleword.

Table 3-3
Device-Specific Functions for the RX01 Disk Driver

QIO$C IO.RPB,...,<stadd,size,,,pbn> Read physical block

QIO$C IO.WDD,...,<stadd,size,,,pbn> Write physical block (with deleted data mark)

QIO$C IO.WPB,...,<stadd,size,,,pbn> Write physical block

where: stadd is the starting address of the data buffer (must be on a word boundary).

size is the data buffer size in bytes must be even and greater than zero).

pbn is the physical block number where the transfer starts (must be in the range 0 to 2001.).

3.4 STATUS RETURNS

The error and status conditions listed in Table 3-4 are returned by the disk drivers described in this chapter.

Table 3-4
Disk Status Returns

Code	Reason
IS.SUC	Successful completion The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading or writing.
IS.PND	I/O request pending The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with zeros.
IS.RDD	Deleted data mark read A deleted record was encountered while executing an IO.RPB function. The second word of the I/O status block can be examined to determine the number of bytes processed.
IE.ALN	File already open The task attempted to open a file on the physical device unit associated with specified LUN, but a file has already been opened by the issuing task on that LUN.

Table 3-4 (Cont.)
Disk Status Returns

Code	Reason
IE.BLK	Illegal block number An illegal logical block number was specified. This code would be returned, for example, if block 4800 were specified for an RK05 disk, on which legal block numbers extend from zero through 4799.
IE.BYT	Byte-aligned buffer specified Byte alignment was specified for a buffer, but only word alignment is legal for disk. Alternatively, the length of a buffer is not an appropriate number of bytes. For example, all RP03 and RP02 disk transfers must be multiples of four bytes.
IE.DNR	Device not ready The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation.
IE.IFC	Illegal function A function code was specified in an I/O request that is illegal for disks.
IE.NLN	File not open The task attempted to close a file on the physical device unit associated with the specified LUN, but no file was currently open on that LUN.
IE.NOD	Insufficient buffer space Dynamic storage space has been depleted, and there was insufficient buffer space available to allocate a secondary control block. For example, if a task attempts to open a file, buffer space for the window and file control block must be supplied by the Executive. This code is returned when there is not enough space for this operation.
IE.OFL	Device off-line The physical device unit associated with the LUN specified in the QIO directive was not on-line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.OVR	Illegal read overlay request A read overlay was requested, and the physical device unit specified in the QIO directive was not the physical device unit from which the task was installed. The read overlay function can only be executed on the physical device unit from which the task image containing the overlays was installed.

Table 3-4 (Cont.)
Disk Status Returns

Code	Reason
IE.PRI	Privilege violation The task which issued the request was not privileged to execute that request. For disk, this code is returned if a nonprivileged task attempts to read or write a mounted volume directly (i.e., using IO.RLB or IO.WLB). Also, this code is returned if any task attempts to attach a mounted volume.
IE.SPC	Illegal address space The buffer specified for a read or write request was partially or totally outside the address space of the issuing task. Alternately, a byte count of zero was specified.
IE.VER	Unrecoverable error After the system's standard number of retries has been attempted upon encountering an error, the operation still could not be completed. For disk, unrecoverable errors are usually parity errors.
IE.WLK	Write-locked device The task attempted to write on a disk that was physically write-locked.

When a disk I/O error condition is detected, an error is usually not returned immediately. Instead, RSX-11M attempts to recover from most errors by retrying the function as many as eight times. Unrecoverable errors are generally parity, timing, or other errors caused by a hardware malfunction.

CHAPTER 4

DECTAPE DRIVER

4.1 INTRODUCTION

The RSX-11M DECtape driver supports the TC11-G dual DECtape controller with up to three additional dual DECtape transports. The TC11-G is a dual-unit, bidirectional, magnetic-tape transport system for auxiliary data storage. DECtape is formatted to store data at fixed positions on the tape, rather than at unknown or variable positions as on coventional magnetic tape. The system uses redundant recording of the mark, timing, and data tracks to increase reliability. Each reel contains 578 logical blocks. As with disk, each of these blocks can be accessed separately, and each contains 256 words.

4.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the GET LUN INFORMATION system directive (the first characterics word) contains the following information for DECtapes. A bit setting of 1 indicates that the described characteristic is true for DECtapes.

Bit	Setting	Meaning
0	0	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	1	Directory device
4	0	Single-directory device
5	0	Sequential device
6-12	0	Reserved
13	0	Device mountable as a communications channel
14	1	Device mountable as a FILES-11 volume
15	1	Device mountable

Words 3 and 4 of the buffer are undefined; word 5 indicates the default buffer size, for DECtape 512 bytes.

4.3 QIO MACRO

This section summarizes standard and device-specific QIO functions for the DECtape driver.

4.3.1 Standard QIO Functions

Table 4-1 lists the standard functions of the QIO macro that are valid for DECtape.

Table 4-1
Standard QIO Functions for DECtape

Format	Function
QIO$C IO.ATT,...	Attach device*
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Not applicable (NOP)
QIO$C IO.RLB,...,<stadd,size,,,lbn>	Read logical block (forward)
QIO$C IO.RVB,...,<stadd,size,,,lbn>	Read virtual block (forward)
QIO$C IO.WLB,...,<stadd,size,,,lbn>	Write logical block (forward)
QIO$C IO.WVB,...,<stadd,size,,,lbn>	Write virtual block (forward)

where: stadd is the starting address of the data buffer (must be on a word boundary).

size is the data buffer size in bytes (must be even and greater than zero).

lbn is the logical block number on the DECtape where the transfer starts (must be in the range 0-577).

IO.RVB and IO.WVB are associated with file operations (see the RSX-11M I/O Operations Reference Manual). For these functions to be executed, a file must be open on the specified LUN if the volume associated with the LUN is mounted. Otherwise, the virtual I/O request is converted to a logical I/O request using the specified block numbers.

4.3.2 Device-Specific QIO Functions

The device-specific functions of the QIO macro that are valid for DECtape are shown in Table 4-2.

---

*Only unmounted volumes may be attached. An attempt to attach a mounted volume will result in an IE.PRI status being returned in the I/O status doubleword.

Table 4-2
Device-Specific Functions for DECtape

Format	Function
QIO$C IO.RLV,...,<stadd,size,,,lbn>	Read logical block (reverse)
QIO$C IO.WLV,...,<stadd,size,,,lbn>	Write logical block (reverse)

Where: stadd is the starting address of the data buffer (must be on a word boundary).

size is the data buffer size in bytes (must be even and greater than zero).

lbn is the logical block number on the DECtape where the transfer starts (must be in the range 0-577).

4.4 STATUS RETURNS

The error and status conditions listed in Table 4-3 are returned by the DECtape driver described in this chapter.

Table 4-3
DECtape Status Returns

Code	Reason
IS.SUC	Successful completion The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading or writing.
IS.PND	I/O request pending The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with zeros.
IE.ALN	File already open The task attempted to open a file on the physical device unit associated with the specified LUN, but a file has already been opened by the issuing task on that LUN.
IE.BLK	Illegal block number. An illegal logical block number was specified for DECtape. The number exceeds 577 (1101 octal).

Table 4-3 (Cont.)
DECtape Status Returns

Code	Reason
IE.BYT	Byte-aligned buffer specified. Byte alignment was specified for a buffer, but only word alignment is legal for DECtape. Alternately, the length of the buffer is not an even number of bytes.
IE.DNR	Device not ready The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation.
IE.IFC	Illegal function A function code was specified in an I/O request that is illegal for DECtape.
IE.NLN	File not open The task attempted to close a file on the physical device unit associated with the specified LUN, but no file was currently open on that LUN.
IE.NOD	Insufficient buffer space Dynamic storage space has been depleted, and there was insufficient buffer space available to allocate a secondary control block. For example, if a task attempts to open a file, buffer space for the window and file control block must be supplied by the Executive. This code is returned when there is not enough space for this operation.
IE.OFL	Device off-line The physical device unit associated with the LUN specified in the QIO directive was not on-line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.OVR	Illegal read overlay request A read overlay was requested and the physical device unit specified in the QIO directive was not the physical device unit from which the task was installed. The read overlay function can only be executed on the physical device unit from which the task image containing the overlays was installed.
IE.PRI	Privilege violation The task which issued the request was not privileged to execute that request. For DECtape, this code is returned when a nonprivileged task attempts to read or write a mounted volume directly (i.e., IO.RLB, IO.RLV, IO.WLB, or IO.WLV). Also, this code is returned if any task attempts to attach a mounted volume.

Table 4-3 (Cont.)
DECtape Status Returns

Code	Reason
IE.SPC	Illegal address space The buffer specified for a read or write request was partially or totally outside the address space of the issuing task. Alternately, a byte count of zero was specified.
IE.VER	Unrecoverable error After the system's standard number of retries has been attempted upon encountering an error, the operation still could not be completed. For DECtape, this code is returned to indicate any of the following conditions. . A parity error was encountered. . The task attempted a forward multi-block transfer past block 577 (1101 octal). . The task attempted a backward multi-block transfer past block zero.
IE.WLK	Write-locked device The task attempted to write on a DECtape unit that was physically write-locked.

4.4.1 DECtape Recovery Procedures

When a DECtape I/O error condition is detected, RSX-11M attempts to recover from the condition by retrying the function as many as five times. Unrecoverable errors are generally parity, mark track, or other errors caused by a faulty recording medium or a hardware malfunction. An unrecoverable error condition also occurs when a read or write operation is performed past the last block of the DECtape on a forward operation, or the first block of the DECtape on a reverse operation.

In addition to the standard error conditions, an unrecoverable error is reported when the "rock count" exceeds eight. The rock count is the number of times the DECtape driver reverses the direction of the tape while looking for a block number. Assume that the block numbers on a portion of DECtape are 99, 96, and 101, where one bit was dropped from block number 100, making it 96. If an I/O request is received for block 100 and the tape is positioned at block 99, the driver starts searching forward for block 100. The first block to be encountered is 96 and because the driver is searching for block 100 in a forward direction and 96 is less than 100, the search continues forward. Block 101 is the next block, and because number 101 is greater than 100, the driver reverses the direction of the tape and

starts to search backward. The next block number in this direction is 96 and direction is reversed again, because 100 is greater than 96. To prevent the DECtape from being hung in this position, continually rocking between block numbers 96 and 100, a maximum rock count of eight has been established.

4.4.2 Select Recovery

If the DECtape unit is in an off-line condition when the I/O function is performed, the message shown below is output on the operator's console.

```
*** DTn:  -- SELECT ERROR
```

where n is the unit number of the drive that is currently off-line. The user should respond by placing the unit to REMOTE. The driver retries the function, from the beginning, once every second. It displays the message once every 15 seconds until the appropriate DECtape unit is selected. A select error may also occur when there are two drives with the same unit number or when no drive has the appropriate unit number.

4.5 PROGRAMMING HINTS

This section contains information on important programming considerations relevant to users of the DECtape driver described in this chapter.

4.5.1 DECtape Transfers

If the transfer length on a write is less than 256 words, a partial block is transferred with zero fill for the rest of the physical block. If the transfer length on a read is less than 256 words, only the number of words specified is transferred. If the transfer length is greater than 256 words, more than one physical block is transferred.

4.5.2 Reverse Reading and Writing

The DECtape driver supports reverse reading and writing, because these functions speed up data transfers in some cases. A block should normally be read in the same direction in which it was written. If a block is read from a DECtape into memory in the opposite direction from that in which it was written, it is reversed in memory (e.g., word 255 becomes word 0, and 254 becomes word 1). If this occurs, the user must then reverse the data within memory.

4.5.3 Speed Considerations When Reversing Direction

It is possible to reverse direction at any time while reading or writing DECtape. However, the user should understand that reversing

direction substantially slows down the movement of the tape. Because DECtape must be moving at a certain minimum speed before reading or writing can be performed, a tape block cannot be accessed immediately after reversing direction. Two blocks must be bypassed before a read or write function can be executed, to give the tape unit time to build up to normal access speed. Furthermore, when a request is issued to read or write in a certain direction, the tape first begins to move in that direction, then starts detecting block numbers. The following examples illustrate these principles.

If a DECtape is positioned at block number 12 and the driver receives a request to read block 10 forward, the tape starts to move forward, in the direction requested. When block number 14 is encountered, the driver reverses the direction of the tape, since 14 is greater than 10. The search continues backward, and block numbers 11 and 10 are encountered. Because the direction must be reversed and the driver requires two blocks to build up sufficient speed for reading, block number 9 and 8 are also bypassed in the backward direction. Then the direction is reversed and the driver encounters blocks 8 and 9 forward before reaching block number 10 and executing the read request.

4.5.4 Aborting a Task

If a task is aborted while waiting for a unit to be selected, the DECtape driver recognizes this fact within one second.

CHAPTER 5

MAGNETIC TAPE DRIVERS

5.1 INTRODUCTION

RSX-11M supports three magnetic tape devices: the TU10, the TS03, and the TU16. Table 5-1 summarizes these devices and subsequent sections describe them in greater detail.

Table 5-1
Standard Magtape Devices

	TU10	TU16	TS03
Number of channels	7 or 9	9	9
Recording density, in frames per inch	For 7-channel: 200, 556, or 800; for 9-channel: 800	800 or 1600	800
Tape speed, in inches per second	45	45	15
Maximum data transfer rate, in bytes per second	36,000	For 800 bpi: 36,000; for 1600 bpi: 72,000	12,000
Recording Method	NRZI	NRZI or Phase Encoding	NRZI

Programming for Magtape is quite similar to programming for the magnetic tape cassette (see Chapter 6). Unlike cassette, however, Magtape can handle variable-length records and allows the user to select a parity mode.

RSX-11M does not support a file structure for Magtape.

5.1.1 TU10/TS03 Magnetic Tape

The TU10/TS03 consists of a TM11 controller with a TU10 or TS03 transport. It is a low-cost, high performance system for serial

storage of large volumes of data and programs in an industry-compatible format. All recording is non-return-to-zero, inverted (NRZI).

5.1.2 TU16 Magnetic Tape

The TU16 consists of an RH11/RH70 controller, a TM02 formatter, and a TU16 transport. It is quite similar to the TU10 but is a Massbus device, with a common controller, a specialized formatter, and a drive. Recording is either 800 bpi NRZI or 1600 bpi phase-encoded (PE).

5.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the GET LUN INFORMATION system directive (the first characteristics word) contains the following information for Magtapes. A bit setting of 1 indicates that the described characteristic is true for Magtapes.

Bit	Setting	Meaning
0	1	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	0	Directory device
4	0	Single-directory device
5	1	Sequential device
6-12	0	Reserved
13	0	Device mountable as a communications channel
14	0	Device mountable as a FILES-11 volume
15	0	Device mountable

Words 3 and 4 of the buffer are undefined; word 5 indicates the default buffer size, for Magtapes 512 bytes.

5.3 QIO MACRO

This section summarizes standard and device-specific QIO functions for the Magtape drivers.

5.3.1 Standard QIO Functions

Table 5-2 lists the standard functions of the QIO macro that are valid for Magtape.

Table 5-2
Standard QIO Functions for Magtape

Format	Function
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O requests
QIO$C IO.RLB,...,<stadd,size>	Read logical block (read tape into buffer)
QIO$C IO.RVB,...,<stadd,size>	Read virtual block (read tape into buffer)
QIO$C IO.WLB,...,<stadd,size>	Write logical block (write buffer contents to tape)
QIO$C IO.WVB,...,<stadd,size>	Write virtual block (write buffer contents to tape)

where: stadd is the starting address of the data buffer (must be on a word boundary).

size is the data buffer size in bytes (must be even, greater than zero, and, for a write, must be at least 14 bytes).

IO.KIL does not cancel an in progress request unless a select error has occurred.

5.3.2 Device-Specific QIO Functions

Table 5-3 lists the device-specific functions of the QIO macro that are valid for Magtape. Additional details on certain functions appear below.

Table 5-3
Device-Specific QIO Functions for Magtape

Format	Function
QIO$C IO.EOF,...	Write end-of-file mark (tape mark)
QIO$C IO.RWD,...	Rewind unit
QIO$C IO.RWU,...	Rewind and turn unit off-line
QIO$C IO.SEC,...	Sense tape characteristics
QIO$C IO.SMO,...,<cb>	Mount tape and set tape characteristics
QIO$C IO.SPB,...,<nbs>	Space blocks
QIO$C IO.SPF,...,<nes>	Space files
QIO$C IO.STC,...,<cb>	Set tape characteristics

where: cb represents the characteristic bits to set.

nbs is the number of blocks to space past (positive if forward, negative if reverse).

nes is the number of EOF marks to space past (positive if forward, negative if reverse).

5.3.2.1 IO.RWD - Completion of IO.RWD means that the rewind has been initiated. However, a request for the same unit will be queued by the driver until load point (BOT) is reached.

5.3.2.2 IO.RWU - IO.RWU is normally used when operator intervention is required (e.g., to load a new tape). The operator must turn the unit back on-line manually before subsequent operations can proceed.

5.3.2.3 IO.SEC - This function returns the tape characteristics in the second I/O status word. The tape characteristic bits are defined as follows:

Bit	Meaning When Set	Can Be Set by IO.SMO and IO.STC
0	For TU10, 556 bpi density (seven-channel). For TU16, reserved.	X
1	For TU10, 200 bpi density (seven-channel). For TU16, reserved.	X

Bit	Meaning When Set	Can Be Set by IO.SMO and IO.STC
2	For TU10, core-dump mode (seven-channel, see below). For TU16, reserved.	X
3	Even parity (default is odd).	X
4	Tape is past EOT.	
5	Last tape command encountered EOF (unless last command was backspace).	
6	Writing is prohibited.	X
7	Writing with extended inter-record gap is prohibited (i.e., no recovery is attempted after write error).	X
8	Select error on unit (reserved for driver; always 0 when read by user).	
9	Unit is rewinding (reserved for driver; always 0 when read by user).	
10	Tape is physically write-locked.	
11	For TU10/TS03, reserved. For TU16, 1600 bpi, density.	X
12	For TU10, drive is seven-channel. For TU16, reserved.	
13	Tape is at load point (BOT).	
14	Tape is at end-of-volume (EOV).	
15	Tape is past EOV (reserved for driver; always 0 when read by user).	

In core-dump mode (TU10 only, 800 bpi density, and seven-channel), each eight-bit byte is written on two tape frames, four bits per frame. In other modes on seven-channel tape, only six low-order bits per byte are written.

The effect of these settings is illustrated in Figure 5-1 for the TU10 and in Figure 5-2 for TU16.

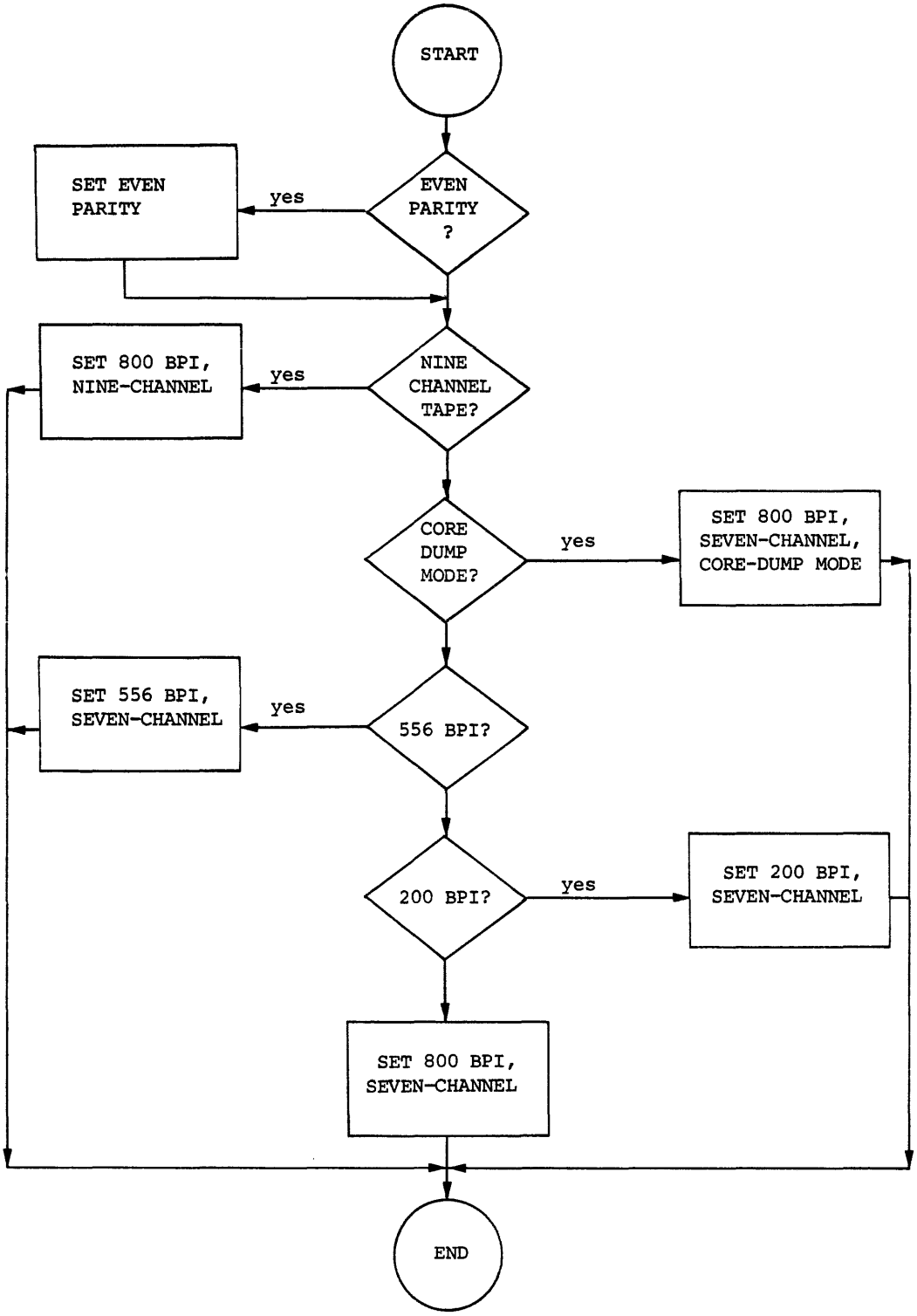


Figure 5-1
Determination of Tape Characteristics
for the TU10

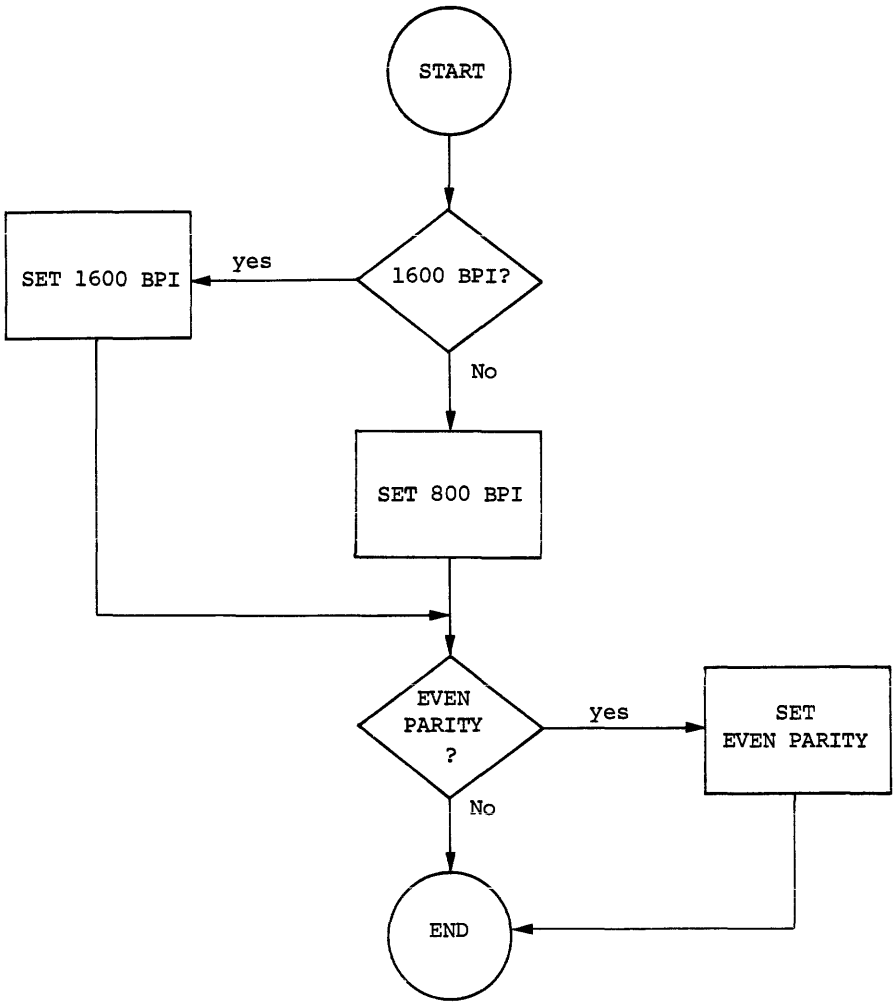


Figure 5-2
Determination of Tape Characteristics
for the TU16

5.4 STATUS RETURNS

The error and status conditions listed in Tabel 5-4 are returned by the Magtape drivers described in this chapter.

Table 5-4
Magtape Status Returns

Code	Reason
IS.SUC	Successful completion The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if operation involved reading or writing. This code is also returned if nbs equals zero in an IO.SPB function or if nes equals zero in an IO.SPF function.
IS.PND	I/O request pending The operation specified in the QIO directive has not yet been completed. The I/O status block is filled with zeros.
IE.ABO	Operation aborted The specified I/O operation was cancelled via IO.KIL while in progress or while still in the I/O queue.
IE.BBE	Bad block A bad block was encountered while reading or writing and the error persists after nine retries. The number of bytes transferred is returned in the second word of the I/O status block. For TM11, IE.BBE may also indicate that a bad tape error (BTE) has been encountered while reading or spacing.
IE.BYT	Byte-aligned buffer specified Byte alignment was specified for a buffer, but only word alignment is legal for Magtape. Alternatively, the length of a buffer is not an even number of bytes.
IE.DAA	Device already attached The physical device unit specified in an IO.ATT function was already attached by the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task.
IE.DAO	Data overrun On a read, a record exceeded the stated buffer size. The final portion of the buffer is checked for parity, but is not read into memory.

Table 5-4 (Cont.)
Magtape Status Returns

Code	Reason
IE.DNA	Device not attached The physical device unit specified in an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.
IE.DNR	Device not ready The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation. This code is returned to indicate one of the following conditions: . A timeout occurred on the physical device unit (i.e., an interrupt was lost). . A vacuum failure occurred on the Magtape drive. . While trying to read or space, the driver detected blank tape. . The "LOAD" switch on the physical drive was switched to the off position.
IE.EOF	End-of-file encountered An end-of-file (tapemark) was encountered.
IE.EOT	End-of-tape encountered The end-of-tape (physical end-of-volume) was encountered while the tape was moving in the forward direction. A ten-foot length of tape is provided past EOT to be used for writing data and markers, such as volume trailer labels. The IE.EOT code will continue to be returned in the I/O status block until the EOT marker is passed in the reverse direction.
IE.EOV	End-of-volume encountered On a forward spacing function, the logical end-of-volume was encountered. An end-of-volume is two consecutive end-of-file marks (EOF), or a beginning-of-tape mark (BOT) followed by an EOF. The tape is normally left positioned between the two marks.
IE.FHE	Fatal hardware error Nonrecoverable hardware malfunction.
IE.IFC	Illegal function A function code was specified in an I/O request that is illegal for Magtape.

Table 5-4 (Cont.)
Magtape Status Returns

Code	Reason
IE.OFL	Device off-line The physical device unit associated with the LUN specified in the QIO directive was not on-line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.SPC	Illegal address space The buffer specified for a read or write request was partially or totally outside the address space of the issuing task. For Magtape, this code is also returned if a byte count of zero was specified or if the user attempted to write a block that was less than 14 bytes long.
IE.VER	Unrecoverable error After the system's standard number of retries has been attempted upon encountering an error, the operation still could not be completed. For Magtape, this code is returned in the case of CRC or checksum errors or when a tape block could not be read.
IE.WLK	Write-locked device The task attempted to write on a Magtape unit that was physically write-locked. Alternately, tape characteristic bit 6 was set by the software to write-lock the unit logically.

After read and write functions, the second I/O status word contains the number of bytes actually processed by the function. After spacing functions, it contains the number of blocks of files spaced over. The EOF mark counts as one block. If an EOF mark is encountered by a read operation, the second I/O status word will contain an octal two.

5.4.1 Select Recovery

If a request fails because the desired unit is off-line, no drive has the desired unit number, or has its power off, the following message is output on the operator's console:

```
*** MTn: -- SELECT ERROR
```

Where n is the unit number of the specified drive. The driver checks the unit for readiness and repeats the message every 15 seconds until the requesting task is aborted or the unit is made available. In the latter case, the driver then proceeds with the request.

5.4.2 Retry Procedures for Reads and Writes

If an error occurs during a read (e.g., vertical parity error), the recovery procedure depends on the type of magtape in use. A bad tape error on a TU10/TS03 results in an immediate return of the error code IE.BBE. All other read errors for both the TU10/TS03 and TU16 are retried by backspacing one record and then rereading the record in question. If the error persists after nine retries, IE.BBE is returned.

Write recovery is the same for both the TU10/TS03 and TU16. When a write operation fails, the driver attempts to avoid the bad spot on the tape by means of an extended interrecord gap (IRG). This means that it backspaces, makes the IRG just before the record three inches longer, and then retries the write. If the error persists after nine retries, IE.BBE is returned. The requesting task can use IO.STC to prohibit writing with an extended interrecord gap. In this case, IE.BBE is returned as soon as a write fails.

5.5 PROGRAMMING HINTS

This section contains information on important programming considerations relevant to users to Magtape drivers described in this chapter.

5.5.1 Block Size

Each block must contain an even number of bytes, at least 14 for a write and at most 65,534. It is more reasonable, however, to work with a block size of approximately 2,048 bytes.

5.5.2 Importance of Resetting Tape Characteristics

A task that uses Magtape should always set the tape characteristics to the proper value before beginning I/O operations. The task cannot be certain in what state a previous task left these characteristics. It is also possible that an operator might have changed the Magtape unit selection. If the selection switch is changed, the new physical device unit may not correspond to the characteristics of the unit described by the respective unit control block.

5.5.3 Aborting a Task

If a task is aborted while waiting for a Magtape unit to be selected, the Magtape driver recognizes this fact within one second.

5.5.4 Writing an Even-Parity Zero

If an even-parity zero were written normally, it would appear to the drive as blank tape. It is therefore converted to 20 (octal). If this conversion is undesirable, the user must ensure that no even-parity zeros are output on the tape.

CHAPTER 6

CASSETTE DRIVER

6.1 INTRODUCTION

RSX-11M supports the TA11 magnetic tape cassette (a TA11 controller with a TU60 dual transport). Programming for cassette is quite similar to programming for Magtape (see Chapter 5). The TA11 system is a dual-drive, reel-to-reel unit designed to replace paper tape. Its two drives run nonsimultaneously, using Digital Proprietary Philips-type cassettes.

The maximum capacity of a cassette, in bytes, is 92,000 (minus 300 per file gap and 46 per interrecord gap). It can transfer data at speeds of up to 562 bytes per second. Recording density ranges from 350 to 700 bits ber inch, depending on tape postion.

6.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the GET LUN INFORMATION system directive (the first characteristics word) contains the following information for cassettes. A bit setting of 1 indicates that the described characteristic is true for cassettes.

Bit	Setting	Meaning
0	1	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	0	Directory device
4	0	Single-directory device
5	1	Sequential device
6-12	0	Reserved
13	0	Device mountable as a communications channel
14	0	Device mountable as a FILES-11 volume
15	0	Device mountable

Words 3 and 4 of the buffer are undefined; word 5 indicates the default buffer size, for cassettes 128 bytes.

6.3 QIO MACRO

This section summarizes standard and device-specific QIO functions for the cassette driver.

6.3.1 Standard QIO Functions

Table 6-1 lists the standard functions of the QIO macro that are valid for cassette.

Table 6-1
Standard QIO Functions for Cassette

Format	Function
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O requests
QIO$C IO.RLB,...,<stadd,size>	Read logical block (read tape into buffer)
QIO$C IO.RVB,...,<stadd,size>	Read virtual block (read tape into buffer)
QIO$C IO.WLB,...,<stadd,size>	Write logical block (write buffer contents to tape)
QIO$C IO.WVB,...,<stadd,size>	Write virtual block (write buffer contents to tape)

where: stadd is the starting address of the data buffer (may be on a byte boundary).

size is the data buffer size in bytes (must be greater than zero).

IO.KIL does not affect in progress-requests.

6.3.2 Device-Specific QIO Functions

Table 6-2 lists the device-specific functions of the QIO macro that are valid for cassette. The section on programming hints below provides more detailed information about certain functions.

Table 6-2
Device-Specific QIO Functions for Cassette

Format	Function
QIO$C IO.EOF,...	Write end-of-file gap
QIO$C IO.RWD,...	Rewind unit
QIO$C IO.SPB,...,<nbs>	Space blocks
QIO$C IO.SPF,...,<nes>	Space files

where: nbs is the number of blocks to space past (positive if forward, negative if reverse).

nes is the number of EOF gaps to space past (positive if forward, negative if reverse).

6.4 STATUS RETURNS

The error and status conditions listed in Table 6-3 are returned by the cassette driver described in this chapter.

Table 6-3
Cassette Status Returns

Code	Reason
IS.SUC	Successful completion The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading or writing, or the number of blocks or files spaced, if the operation involved spacing blocks or files.
IS.PND	I/O request pending The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with zeros.
IE.ABO	Operation aborted The specified I/O operation was cancelled via IO.KIL while still in the I/O queue.
IE.DAA	Device already attached The physical device unit specified in an IO.ATT function was already attached by the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task.

Table 6-3 (Cont.)
Cassette Status Returns

Code	Reason
IE.DAO	Data overrun The driver was not able to sustain the data rate required by the TA11 controller.
IE.DNA	Device not attached The physical device unit specified by an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.
IE.DNR	Device not ready The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation. This code is returned to indicate one of the following conditions: . The cassette has not been physically inserted. . The unit is off-line. . A timeout occurred on the physical device unit (i.e., an interrupt was lost).
IE.EOF	End-of-file encountered An end-of-file gap was recognized on the cassette tape. This code is returned if an EOF gap is encountered during a read or if the cassette is physically removed during an I/O operation.
IE.EOT	End-of-tape encountered While reading or writing, clear trailer at end-of-tape (EOT) was encountered. Unlike Magtape, writing beyond EOT is not permitted on cassettes. This condition is always sensed on a write before it would be sensed on a read of the same section of tape. If IE.EOT is returned during a write, the cassette head has encountered EOT before finishing the writing of the last block. It is recommended that the user rewrite the block on another cassette in its entirety.
IE.IFC	Illegal function A function code was specified in an I/O request that is illegal for cassette.
IE.OFL	Device off-line The physical device unit associated with the LUN specified in the QIO directive was not on-line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.

Table 6-3 (Cont.)
Cassette Status Returns

Code	Reason
IE.SPC	Illegal address space The buffer specified for a read or write request was partially or totally outside the address space of the issuing task. Alternately, a byte count of zero was specified on a transfer.
IE.VER	Nonrecoverable error This code is returned when a block check error occurs (see section 6.6.5). The cyclic redundancy check (CRC), a two-byte value located at the end of each block, is a checksum that is tested during all read operations to ensure that data is read correctly. This is returned if a read request did not specify exactly the number of bytes of data in the record on tape. If a nonrecoverable error is returned, the user may attempt recovery by spacing backward one block and retrying the read operation.
IE.WLK	Write-locked device The task attempted to write on a cassette unit that was physically write-locked. This code may be returned after an IO.WLB, IO.WVB, or IO.EOF function.

6.4.1 Cassette Recovery Procedures

If an error occurs during a read or write operation, the operation should be retried several times. The recommended maximum number of retries is nine for a read and three for a write because each retry involves backspacing, which does not always position the tape in the same place. More than three retries of a write operation may destroy previously written data. For example, to retry a write, it is best to space two blocks in reverse, then space one block forward. This insures the tape is in the proper position to rewrite the block that encountered the error.

After read and write functions, the second I/O status word contains the number of bytes actually processed by the function. After spacing functions, it contains the number of blocks or files actually spaced.

6.5 STRUCTURE OF CASSETTE TAPE

Figure 6-1 illustrates a general structure for cassette tape. A different structure can be employed if the user wishes.

Here the tape consists of blocks of data interspersed with sections of clear tape that serve as leader, trailer, interrecord gaps (IRGs), and end-of-file gaps.

The logical end-of-tape in this case consists of a sentinel label record, rather than the conventional group of end-of-file gaps. Each file must contain at least one block. The size of each block depends upon the number of bytes the user specifies when writing the block.

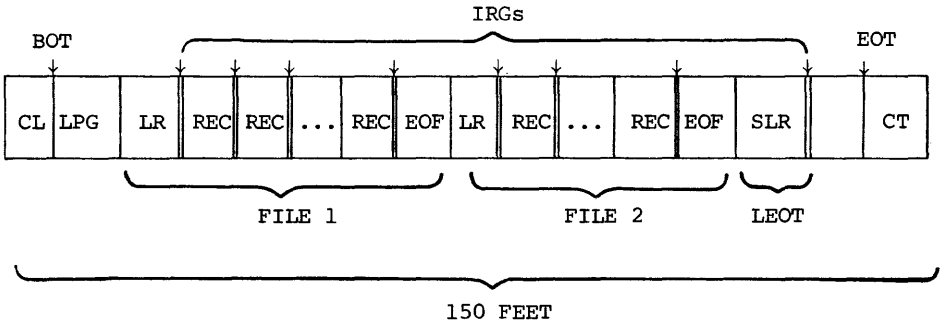


Figure 6-1
Structure of Cassette Tape

Abbreviation	Meaning
CL	Clear leader
BOT	Physical beginning-of-tape
LPG	Load point gap (blank tape written by driver before the first retrievable record)
LR	File label record
REC	Fixed-length record (data)
EOF	End-of-file gap
IRG	Interrecord gap
SLR	Sentinel label record
LEOT	Logical end-of-tape
EOT	Physical end-of-tape
CT	Clear trailer

6.6 PROGRAMMING HINTS

This section contains information on important programming considerations relevant to users of the cassette driver described in this chapter.

6.6.1 Importance of Rewinding

The first cassette operation performed on a tape must always be a rewind to ensure that the tape is positioned to a known place. When it is positioned in clear tape there is no way to determine whether it is in leader at the beginning-of-tape (BOT) or in trailer at the end-of-tape (EOT).

6.6.2 End-of-File and IO.SPF

The hardware senses end-of-file (EOF) as a timeout. When IO.SPF is issued in the forward direction (nes is positive), the tape is positioned two-thirds of the way from the beginning of the final file gap. In effect, this is all the way through the file gap. When IO.SPF is issued in the reverse direction (nes is negative), the tape is positioned one-third of the way from the beginning of the final file gap (i.e., two thirds of the way from the beginning of the last file spaced). Therefore to correctly position the tape for a read or write after issuing IO.SPF in reverse, the user should issue IO.SPB forward for one block, followed by IO.SPB in reverse for one block.

6.6.3 The Space Functions, IO.SPB and IO.SPF

IO.SPB always stops in an IRG, IO.SPF in an EOF gaps. Neither space function actually takes effect until data is encountered. For example, suppose the tape is positioned in clear leader at BOT and the user requests that one block be spaced forward. The drive passes over the remaining leader until it reaches data, passes one block, and stops in the IRG. Similarly, if the same command is issued when the tape is at BOT on a blank tape or a tape containing only EOF gaps, the function does not terminate until EOT.

6.6.4 Verification of Write Operations

Certain errors, such as cyclic redundancy check, are detected on read but not write operations. Therefore, to ensure reliability of recording, it is recommended that the user perform a read as verification of every write operation.

6.6.5 Block Length

The user must specify the exact number of bytes per block when requesting read or write operations. An attempt to read a block with an incorrect byte count causes an unrecoverable error (see section 6.4) to occur.

6.6.6 Logical End-of-Tape

The conventional method of signaling logical end-of-tape by multiple EOF gaps is inadequate for cassettes. This is because multiple EOF gaps are not distinguishable from each other. For example, two sequential EOF gaps would be read as three instead of two. Also spacing functions, since they are triggered by encountering data, can not recognize multiple EOF gaps. Consequently, the use of a sentinel or key record to signal logical end-of-tape is recommended.

CHAPTER 7

LINE PRINTER DRIVER

7.1 INTRODUCTION

The RSX-11M line printer driver supports the line printers summarized in Table 7-1. Subsequent sections of this chapter describe these printers in greater detail.

Table 7-1
Standard Line Printer Devices

Model	Column Width	Character Set	Lines per Minute
LP11-F	80	64	170-1110
LP11-H	80	96	170-1110
LP11-J	132	64	170-1110
LP11-K	132	96	170-1110
LP11-R	132	64	1110
LP11-S	132	96	1110
LP11-V	132	64	300
LP11-W	132	96	300
LS11	132	62	60-200
LV11	132	96	500

7.1.1 LP11 LINE PRINTER DRIVER

The LP11 is a high-speed line printer available in a variety of models. The entire LP11 model line consists of impact printers, using one hammer per column and a revolving drum with upper-case and optional lower-case characters. The LP11-R and LP11-S are fully buffered models which operate at a standard speed of 1110 lines per minute. The other LP11 models have 20-character print buffers. These printers are therefore able to print at full speed if the printed line is no longer than 20 characters. Lines which exceed this maximum are printed at a slower rate. Forms with up to six parts may be used for multiple copies.

7.1.2 LS11 Line Printer

The LS11 is a medium-speed line printer. It has a 20-character print buffer, and lines of 20 characters or less are printed at a rate of 200 lines per minute. Longer lines are printed at a slower rate. RSX-11M does not support the LS11 expanded character set feature.

7.1.3 LV11 Line Printer

The LV11 is a fully-buffered, electrostatic printer-plotter which operates at a standard rate of 500 lines per minute. RSX-11M supports only the LV11 print capability, not the plotter mode.

7.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the GET LUN INFORMATION system directive (the first characteristics word) contains the following information for line printers. A bit setting of 1 indicates that the described characteristic is true for line printers.

Bit	Setting	Meaning
0	1	Record-oriented device
1	1	Carriage-control device
2	0	Terminal device
3	0	Directory device
4	0	Single-directory device
5	0	Sequential device
6-12	0	Reserved
13	0	Device mountable as a communications channel
14	0	Device mountable as a FILES-11 volume
15	0	Device mountable

Words 3 and 4 of the buffer are undefined; word 5 indicates the default size for the device, for line printers the width of the printer carriage (i.e., 80 or 132).

7.3 QIO MACRO

Table 7-2 lists the standard functions of the QIO macro that are valid for line printers.

Table 7-2
Standard QIO Functions for Line Printers

Format	Function
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O requests
QIO$C IO.WLB,...,<stadd,size,vfc>	Write logical block (print buffer contents)
QIO$C IO.WVB,...,<stadd,size,vfc>	Write virtual block (print buffer contents)

where: stadd is the starting address of the data buffer (may be on a byte boundary).

size is the data buffer size in bytes (must be greater than zero).

vfc is a vertical format control character from Table 7-4.

IO.KIL does not cancel an in progress request unless the line printer is in an offline condition because of a power failure or a paper jam or because it is out of paper.

The line printer driver supports no device-specific functions.

7.4 STATUS RETURNS

Table 7-3 lists the error and status conditions that are returned by the line printer driver described in this chapter.

Table 7-3
Line Printer Status Returns

Code	Reason
IS.SUC	Successful completion The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved writing.
IS.PND	I/O request pending The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with zeros.

Table 7-3 (Cont.)
Line Printer Status Returns

Code	Reason
IE.ABO	Operation aborted The specified I/O operation was cancelled while in progress or while in the I/O queue.
IE.DAA	Device already attached The physical device unit specified in an IO.ATT function was already attached by the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task.
IE.DNA	Device not attached The physical device unit specified an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.
IE.IFC	Illegal function A function code was specified in an I/O request that is illegal for line printers.
IE.OFL	Device off-line The physical device unit associated with the LUN specified in the QIO directive was not on-line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.SPC	Illegal address space The buffer specified for a write request was partially or totally outside the address space of the issuing task. Alternately, a byte count of zero was specified.

7.4.1 Ready Recovery

If any of the following conditions occur:

- Paper jam
- Printer out of paper
- Printer turned off-line
- Power failure

the driver determines that the line printer is off-line, and the following message is output on the operator's console:

```
***LPn: -- NOT READY
```

where n is the unit number of the line printer that is not ready. The driver retries the function which encountered the error condition from the beginning, once every second. It displays the message every 15 seconds until the line printer is readied. If a power failure occurs while printing a line, the entire line is reprinted from the beginning when power is restored.

7.5 VERTICAL FORMAT CONTROL

Table 7-4 summarizes the meaning of all characters used for vertical format control on the line printer. Any one of these characters can be specified as the vfc parameter in an IO.WLB or IO.WVB function.

Table 7-4
Vertical Format Control Characters

Octal Value	Character	Meaning
040	blank	SINGLE SPACE: output a line feed, print the contents of the buffer, and output a carriage return. Normally, printing immediately follows the previously printed line.
060	zero	DOUBLE SPACE: output two line feeds, print the contents of the buffer, and output a carriage return. Normally, the buffer contents are printed two lines below the previously printed line.
061	one	PAGE EJECT: output a form feed, print the contents of the buffer, and output a carriage return. Normally, the contents of the buffer are printed on the first line of the next page.
053	plus	OVERPRINT: print the contents of the buffer and perform a carriage return, normally overprinting the previous line.
044	dollar sign	PROMPTING OUTPUT: output a line feed and then print the contents of the buffer.
000	null	INTERNAL VERTICAL FORMAT: the buffer contents are printed without addition of vertical format control characters. In this mode, more than one line of guaranteed contiguous output can be printed per I/O request.

All other vertical format control characters are interpreted as blanks (octal 040).

7.6 PROGRAMMING HINTS

This section contains information on important programming considerations relevant to users of the line printer driver described in this chapter.

7.6.1 RUBOUT Character

The line printer driver discards the ASCII character code 177 during output, because a RUBOUT on the LS11 printer causes a RUBOUT of the hardware print buffer.

7.6.2 Print Line Truncation

If the number of characters to be printed exceeds the width of the print carriage, the driver discards excess characters until it receives one that instructs it to empty the buffer and return to horizontal position 1. The user can determine that truncation will occur by issuing a GET LUN INFORMATION system directive and examining word 5 of the information buffer. This word contains the width of the print carriage in bytes.

7.6.3 Aborting a Task

If a task is aborted while waiting for the line printer to be readied, the line printer driver recognizes this fact within one second.

CHAPTER 8

CARD READER DRIVER

8.1 INTRODUCTION

The RSX-11M card reader driver supports the CR11 card reader. This reader is a virtually jam-proof device which reads EIA standard 80-column punched cards at the rate of 300 per minute. The hopper can hold 600 cards. This device uses a vacuum picker which provides extreme tolerance to damaged cards and makes card wear insignificant. Cards are riffled in the hopper to prevent sticking. The reader uses a strong vacuum to deliver the bottom card. It has a very short card track, so only one card is in motion at a time.

8.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the GET LUN INFORMATION system directive (the first characteristics word) contains the following information for card readers. A bit setting of 1 indicates that the described characteristic is true for card readers.

Bit	Setting	Meaning
0	1	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	0	Directory device
4	0	Single-directory device
5	0	Sequential device
6-12	0	Reserved
13	0	Device mountable as a communications channel
14	0	Device mountable as a FILES-11 volume
15	0	Device mountable

Words 3 and 4 of the buffer are undefined; word 5 indicates the default buffer size, which is 80 bytes for the card reader.

8.3 QIO MACRO

This section summarizes standard and device-specific QIO functions for the card reader driver.

8.3.1 Standard QIO Functions

Table 8-1 lists the standard functions of the QIO macro that are valid for the card reader.

Table 8-1
Standard QIO Functions for the Card Reader

Format	Function
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O requests
QIO$C IO.RLB,...,<stadd,size>	Read logical block (alphanumeric)
QIO$C IO.RVB,...,<stadd,size>	Read virtual block (alphanumeric)

where: stadd is the starting address of the data buffer (may be on a byte boundary).

size is the data buffer size in bytes (must be greater than zero).

IO.KIL does not cancel an in progress request unless the card reader is in an offline condition because of a pick, read, stack or hopper check, because of power failure, or because the RESET button has not been depressed.

8.3.2 Device-Specific QIO Function

The device-specific function of the QIO macro that is valid for the card reader is shown in Table 8-2.

Table 8-2
Device-Specific QIO Function for the Card Reader

Format	Function
QIO$C IO.RDB,...,<stadd,size>	Read logical block (binary)

where: stadd is the starting address of the data buffer (may be on a byte boundary).

size is the data buffer size in bytes (must be greater than zero).

8.4 STATUS RETURNS

There are a wide variety of error conditions and recovery procedures related to the use of the card reader. This section describes the three major ways in which the system reports error conditions.

1. Lights and indicators on the card reader panel are turned on or off to indicate particular operational problems such as read, pick, stack or hopper checks. Switches are available to turn the reader power on and off and to allow the user to reset after correcting an error condition.

2. A message is output on the operator's console if operational checks or power problems occur.

3. An I/O completion code is returned in the low-order byte of the first word of the I/O status block specified in the QIO macro to indicate success or failure on completion of an I/O function.

The following subsections describe each of these returns in detail.

8.4.1 Card Input Errors and Recovery

The table included below describes all external lights and switches used to indicate to the operator that a hardware problem has occurred and must be corrected. There are two classes of hardware errors:

- Those requiring the operator to ready the reader and try the operation again.

- Those requiring the operator to remove the last card from the output stacker, to replace it in the input hopper, and to try the operation again.

In the first case, the card reader was unable to read the current card. In the second, the card was read incorrectly and must be physically removed from the output stacker. The card reader driver automatically restarts a read operation within one second after the cards have been replaced in the input hopper.

Table 8-3 summarizes the functions of lights and indicators on the front panel of the card reader. It discusses common operational errors which might be encountered while reading cards and recovery procedures associated with these error conditions.

Table 8-3
Card Reader Switches and Indicators

Indicator	Description	Action	Recovery
POWER switch	pushbutton indicator switch (alternate action: pressed for both ON and OFF)	Controls application of all power to the card reader. When indicator is off, depressing switch applies power to reader and causes associated indicator to light. When indicator is lit, depressing switch removes all power from reader and causes indicator to go out.	Card may have been read incorrectly; restore power if possible by depressing the POWER switch; insert the card again as the first card in the input hopper, and press the RESET switch; in some cases, it may be necessary to restart the program.
READ CHECK indicator	white light	When lit, this light indicates that the card just read may be torn on the leading or trailing edges, or that the card may have punches in column positions 0 or 81. Because READ CHECK indicates an error condition, whenever this indicator is lit, it causes the card reader to stop operation and extinguishes the RESET indicator.	Card was read incorrectly; duplicate if necessary, insert the card again as the first card in the input hopper and press the RESET switch.

Table 8-3 (Cont.)
Card Reader Switches and Indicators

Indicator	Description	Action	Recovery
PICK CHECK indicator	white light	When lit, this light indicates that the card reader failed to move a card into the read station after it received a READ COMMAND from the controller. Stops card reader operation and extinguishes RESET indicator.	Card could not be read; press the RESET switch to try again or remove the cards from the input hopper, smooth the leading edges, replace, and then press the RESET switch.
STACK CHECK indicator	white light	When lit, this light indicates that the previous card was not properly seated in the output stacker and therefore may be badly mutilated. Stops card reader operation and extinguishes RESET indicator.	Card may have been read incorrectly and is not positioned properly in the output stacker; duplicate the card if it is damaged; insert the card again as the first card in the input hopper and press the RESET switch.
HOPPER CHECK indicator	white light	When lit, this light indicates that either the input hopper is empty or that the output stacker is full.	Card may have been read incorrectly; empty the stacker or fill the hopper; insert the card again as the fist card in the input hopper and press the RESET switch.
STOP switch	momentary pushbutton/ indicator switch (red light)	When depressed, immediately lights and drops the READY line, thereby extinguishing the RESET indicator. Card reader operation then stops as soon as the card currently in the read station has been read. This switch has no effect on the system power; it only stops the current operation.	

Table 8-3 (Cont.)
Card Reader Switches and Indicators

Indicator	Description	Action	Recovery
RESET switch	momentary pushbutton/ indicator switch (green light)	When depressed and released, clears all error flip-flops and initializes card reader logic. Associated RESET indicator lights to indicate that the READY signal is applied to the controller. The RESET indicator goes out whenever the STOP switch is depressed or whenever an error indicator lights (READ CHECK, PICK CHECK, STACK CHECK, or HOPPER CHECK).	

8.4.2 Ready and Card Reader Check Recovery

If any of the following conditions occurs:

- POWER failure
- reset switch not pressed (reader offline)

the driver determines that the card reader is not ready, and the following message is output on the operator's console:

```
*** CRn: -- NOT READY
```

If any of the following conditions occurs:

- Pick error (PICK CHECK)
- Read error (READ CHECK)
- Output stacker error (STACK CHECK)
- Input hopper out of cards (HOPPER CHECK)
- Output stacker full (HOPPER CHECK)

the driver determines that a card reader check has occurred, and the following message is output on the operator's console:

```
*** CRn: -- READ FAILURE. CHECK HARDWARE STATUS
```

where n is the unit number of the card reader that is not ready. The operator should correct the error and press RESET: the driver

attempts the function from the beginning, once every second. It displays the message once every 15 seconds until the card reader is readied. In all cases except pick error, the last card read should be reinserted in the input hopper, as described in section 8.4.1.

8.4.3 I/O Status Conditions

The error and status conditions listed in Table 8-4 are returned by the card reader driver described in this chapter.

Table 8-4
Card Reader Status Returns

Code	Reason
IS.SUC	Successful completion The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading.
IS.PND	I/O request pending The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with zeros.
IE.ABO	Operation aborted The specified I/O operation was cancelled while in progress or while still in the I/O queue.
IE.DAA	Device already attached The physical device unit specified in an IO.ATT function was already attached by the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task.
IE.DNA	Device not attached The physical device unit specified in an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.
IE.EOF	End-of-file encountered An end-of-file control card was recognized.
IE.IFC	Illegal function A function code was specified in an I/O request that is illegal for card readers.

Table 8-4 (Cont.)
Card Reader Status Returns

Code	Reason
IE.NOD	Buffer allocation failure Dynamic storage space has been depleted, and there was insufficient buffer space available to allocate a card buffer (i.e., cards are read into a driver buffer, translated, and then moved to the user buffer).
IE.OFL	Device off-line The physical device unit associated with the LUN specified in the QIO directive was not on-line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.SPC	Illegal address space The buffer specified for a read request was partially or totally outside the address space of the issuing task. Alternately, a byte count of zero was specified.

8.5 FUNCTIONAL CAPABILITIES

The card reader driver can perform the following functions:

1. Read cards in DEC026 format and translate to ASCII.
2. Read cards in DEC029 format and translate to ASCII.
3. Read cards in binary format.

If the QIO macro specifies the IO.RLB or IO.RVB function, the driver interpets all data as alphanumeric (026 or 029 format). As explained below, control characters indicate whether 026 or 029 is desired. If the QIO macro specifies IO.RDB, the driver interprets all data, including 026 and 029 control characters, as binary.

8.5.1 Control Characters

Table 8-5 lists the multipunched cards that the card reader driver recognizes as control characters. They are never transferred to the user's buffer or included in the count of transferred bytes in alphanumeric mode. In binary mode the only control card recognized is binary EOF.

Table 8-5
Card Reader Control Characters

Punches	Columns	Meaning
12-11-0-1-6-7-8-9	1	End-of-file (alphanumeric)
12-11-0-1-6-7-8-9	(all 8 punches in the first 8 columns)	End-of-file (binary)
12-2-4-8	1	026-coded cards follow
12-0-2-4-6-8	1	029-coded cards follow

DEC026 is the default translation mode when the system is bootstrapped. This mode remains in effect until explicitly changed by a control card indicating that DEC029 cards will follow. After encountering a DEC029 control card, the driver translates all cards in DEC029 format unless another DEC026 control card is encountered. This card overrides the 029 mode specification and indicates that subsequent cards are to be translated in 026 format. Control characters are addressed to the card reader itself, and remain in effect even when the reader is attached and subsequently detached.

8.6 CARD READER DATA FORMATS

The card reader reads data in either alphanumeric or binary format.

8.6.1 Alphanumeric Format (026 and 029)

Table 8-6 summarizes the translation from DEC026 or DEC029 card codes to ASCII.

Table 8-6
Translation from DEC026 or DEC029 to ASCII

Character	Non-Parity ASCII	DEC029	DEC026	Character	Non-Parity ASCII	DEC029	DEC026
{	173	12 0	12 0	'	054	0 8 3	0 8 3
}	175	11 0	11 0	-	055	11	11
SPACE	040	none	none	.	056	12 8 3	12 8 3
!	041	12 8 7	12 8 7	/	057	0 1	0 1
"	042	8 7	0 8 5	0	060	0	0
	043	8 3	0 8 6	1	061	1	1
$	044	11 8 3	11 8 3	2	062	2	2
%	045	0 8 4	0 8 7	3	063	3	3
AND	046	12	11 8 7	4	064	4	4
'	047	8 5	8 6	5	065	5	5
(	050	12 8 5	0 8 4	6	066	6	6
)	051	11 8 5	12 8 4	7	067	7	7
*	052	11 8 4	11 8 4	8	070	8	8
+	053	12 8 6	12	9	071	9	9

Table 8-6 (Cont.)
Translation from DEC026 or DEC029 to ASCII

Character	Non-Parity ASCII	DEC029	DEC026	Character	Non-Parity ASCII	DEC029	DEC026
<	072	8 2	11 8 2	M	115	11 4	11 4
=	073	11 8 6	0 8 2	N	116	11 5	11 5
>	074	12 8 4	12 8 6	O	117	11 6	11 6
?	075	8 6	8 3	P	120	11 7	11 7
:	076	0 8 6	11 8 6	Q	121	11 8	11 8
?	077	0 8 7	12 8 2	R	122	11 9	11 9
@	100	8 4	8 4	S	123	0 2	0 2
A	101	12 1	12 1	T	124	0 3	0 3
B	102	12 2	12 2	U	125	0 4	0 4
C	103	12 3	12 3	V	126	0 5	0 5
D	104	12 4	12 4	W	127	0 6	0 6
E	105	12 5	12 5	X	130	0 7	0 7
F	106	12 6	12 6	Y	131	0 8	0 8
G	107	12 7	12 7	Z	132	0 9	0 9
H	110	12 8	12 8	[	133	12 8 2	11 8 5
I	111	12 9	12 9	\	134	0 8 2	8 7
J	112	11 1	11 1	]	135	11 8 2	12 8 5
K	113	11 2	11 2	^ or ↑	136	11 8 7	8 5
L	114	11 3	11 3	_ or ←	137	0 8 5	8 2

8.6.2 Binary Format

In RSX-11M binary format, the data are not packed, but are transferred exactly as read, one card column per word. Because each word has 16 bits and each card column represents only 12, the data from the column are stored in the rightmost 12 bits of the word. The word's remaining four bits contain zeros.

8.7 PROGRAMMING HINTS

This section contains information on important programming considerations relevant to users of the card reader driver described in this chapter. Section 8.4 contains information on operational error-recovery procedures which might be important from a programming point of view.

8.7.1 Input Card Limitation

Only one card can be read with a single QIO macro call. A request to read more than 80 bytes or columns, the length of a single card, does not result in a multiple card transfer. Only 80 columns are processed. It is possible to read fewer than 80 columns of card input with a QIO read function. The user can specify that only the first 10 columns, for example, of each card are to be read.

8.7.2 Aborting a Task

If a task which is waiting for the card reader to be readied is aborted, the card reader driver recognizes this fact within one second.

CHAPTER 9

MESSAGE-ORIENTED COMMUNICATION DRIVERS

9.1 INTRODUCTION

RSX-11M supports a variety of communication line interfaces synchronous and asynchronous, single-line and multiplexers, character-oriented and message-oriented. These are used for terminal communications, remote job entry, multicomputer interfaces, and laboratory and industrial control communications. Communications line interfaces can be roughly divided into two categories:

- Terminal (character-oriented) communications devices
- Multicomputer (message-oriented) communications devices

Chapter 2 describes the character-oriented asynchronous communications line interfaces used primarily for terminal communications. The PDP-11 PERIPHERALS HANDBOOK contains more detail on these devices. This chapter describes in some detail the RSX-11M message-oriented synchronous and asynchronous communication line interfaces. These are used most frequently in multicomputer communications.

Character-oriented communications devices include the DH11, DJ11, DL11-A, DL11-B, DL11-C and DL11-D interfaces. These are asynchronous multiplexers and single-line interfaces which are used almost exclusively for termimal communications. Transfers on all of these interfaces are performed one character at a time. None of the interfaces in this category have drivers of their own (i.e., they are supported via the terminal driver), and none can be accessed directly as RSX-11M devices.

Message-oriented communications line interfaces are used primarily to link two separate but complementary computer systems. One system must serve as the transmitting device and the other as the receiving device. Devices in this category include the synchronous and asynchronous single-line interfaces summarized in Table 9-1.

Table 9-1
Message-Oriented Communication Interfaces

Model	Type	Function
DA11-B	Asynchronous, parallel	Single-line interface
DL11-E	Asynchronous	Single-line interface

Table 9-1 (Cont.)
Message-Oriented Communication Interfaces

Model	Type	Function
DP11	Synchronous	Single-line interface
DQ11	Synchronous	Single-line interface
DU11	Synchronous	Single-line interface

The message-oriented communication line interfaces are used primarily to transfer large blocks of data.

Whereas the character-oriented interfaces can only be accessed indirectly through the terminal driver, the DA11-B, DL11-E, DP11, DQ11 and DU11 allow I/O requests to be queued directly for them. These devices have drivers of their own and can be accessed by means of the logical device names listed in Table 1-1. These names can be used in assigning LUNs via the ASSIGN LUN system directive, at task build or via the REASSIGN MCR command. The following subsections briefly discuss the message-oriented interfaces supported for RSX-11M.

9.1.1 DA11-B Synchronous Line Interface

The DA11-B provides a bit parallel, direct memory access interface between two PDP-11 computer systems. Data transfers are performed a word at a time and are made directly between the memories of the two systems. The maximum transfer rate is 500,000 baud, and is adjustable by the user to match the system configuration requirements. Being a parallel device, the DA11-B does not utilize sync characters. The interface is half-duplex and transfers data in blocks of up to 32K words.

The DA11-B requires two cooperating computers to effect a data transfer. In order to control the physical link between the computers, the device driver contains its own simple line protocol. This protocol requires one system to issue a receive QIO and the other to issue a transmit QIO before any data is actually transferred.

9.1.2 DL11-E Asynchronous Line Interface

The DL11-E is an asynchronous, serial-bit, single-line interface. It is a block-transfer device used for remote terminal and multicomputer communications. Baud rates are selectable between 50 and 9600, and full data set control is supported.

9.1.3 DP11 Synchronous Line Interface

The DP11 provides a program interrupt interface between a PDP-11 and a serial synchronous line. This interface facilitates the use of the PDP-11 in remote batch processing, remote data collection, and remote concentration applications. The modem control feature allows the DP11 to be used in switched or dedicated configurations.

On the DP11, baud rates are selectable between 2000 and 19,200. The programmer can select a specific sync character which is used to synchronize the transmitting and receiving systems.

9.1.4 DQ11 Synchronous Line Interface

The DQ11 provides a direct memory access interface between a PDP-11 and a serial synchronous line. The direct memory access characteristic of the DQ11 allows the device to operate at speeds higher than those of program interrupt devices, and with a lower interrupt overhead. Modem control of the DQ11 allows the device to be used in switched or dedicated configurations.

The DQ11 handles data rates from 2400 baud to 1,000,000 baud. The limiting rate is determined by the modem and data set interface level converters.

The DQ11 sync character is programmable in the same manner as the DP11 and the DU11. The maximum data block length transmitted is 65,536 characters.

9.1.5 DU11 Synchronous Line Interface

The DU11 synchronous line interface is a single-line communications device which provides a program-controlled interface between the PDP-11 and a serial synchronous line. The PDP-11 can be interfaced with a high-speed line to perform remote batch processing, remote data collection, and remote concentration applications. Modem control is a standard feature of the DU11 and allows the device to be used in switched or dedicated configurations. The DU11 transmits data at a maximum rate of 9600 baud; this rate is limited by modem and data set interface level converters.

The DU11 can be programmed to accept any user-defined sync character. The use of the sync character is the same for the DU11 and the DP11.

9.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the GET LUN INFORMATION system directive (the first characteristics word) contains the following information for message-oriented communication interfaces. A bit setting of 1 indicates that the described characteristic is true for the interfaces described in this chapter.

Bit	Setting	Meaning
0	0	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	0	Directory device

Bit	Setting	Meaning
4	0	Single-directory device
5	0	Sequential device
6-12	0	Reserved
13	1	Device mountable as a communications channel
14	0	Device mountable as a FILES-11 volume
15	1	Device mountable

Words 3 and 4 are undefined, and word 5 has a special meaning for the DL11-E, DQ11, DP11, and the DU11 interfaces. Byte 0 of word 5 contains the number of sync characters to be transmitted before a synching message (e.g., after line turn around in half-duplex operation), and byte 1 is used as a sync counter.

9.3 QIO MACRO

This section summarizes the standard and device-specific functions of the QIO macro that are valid for the communication interfaces described in this chapter.

9.3.1 Standard QIO Functions

Table 9-2 lists the standard functions of the QIO macro that are valid for the communication devices.

Table 9-2
Standard QIO Functions for Communication Interfaces

Format	Function
QIO$C IO.ATT,...	Attach device*
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O requests
QIO$C IO.RLB,...,<stadd,size>	Read logical block (stripping sync)
QIO$C IO.WLB,...,<stadd,size>	Write logical block (preceded by syncs)

where: stadd is the starting address of the data buffer (may be on a byte boundary).

size is the data buffer size in bytes (must be greater than zero).

*Only unmounted channels may be attached. An attempt to attach a mounted channel will result in an IE.PRI status being returned in the I/O status doubleword.

9.3.2 Device-Specific QIO Functions

The specific functions of the QIO macro that are valid for the communication line interfaces are shown in Table 9-3.

Table 9-3
Device-Specific QIO Functions for Communication Interfaces

Format	Function
QIO$C IO.FDX	Set device to full-duplex mode
QIO$C IO.HDX,...	Set device to half-duplex mode
QIO$C IO.INL,...	Initialize device and set device characteristics
QIO$C IO.RNS,...,<stadd,size>	Read logical block, without stripping sync characters (transparent mode). Not applicable to DA11-B or DQ11
QIO$C IO.SYN,...,<syn>	Specify sync character
QIO$C IO.TRM,...	Terminate communication, disconnecting from physical channel
QIO$C IO.WNS,...,<stadd,size>	Write logical block without preceding sync characters (transparent mode). Not applicable to DA11-B.

where: stadd is the starting address of the data buffer (may be on a byte boundary).

size is the data buffer size in bytes (must be greater than zero).

syn is the sync character, expressed as an octal value.

The device-specific functions listed in Table 9-3 are described in greater detail below.

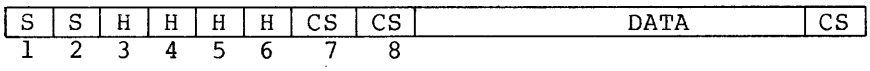
9.3.2.1 IO.FDX - The IO.FDX QIO function is used to set the mode on a DL11-E, DP11, DQ11, or DU11 unit to full-duplex. The IO.FDX function code can be combined (CRed) with the IO.SYN function code, if desired, to set the operational characteristics of the physical device unit.

9.3.2.2 IO.HDX - The IO.HDX QIO function is used to set the mode on a DL11-E, DP11, DQ11, or DU11 unit to half-duplex. The IO.HDX function code can be combined (ORed together) with the IO.SYN function code, if desired, to set the operational characteristics of the physical device unit.

9.3.2.3 IO.INL and IO.TRM - These two QIO functions have the same function code but different modifier bits. IO.INL is used to

initialize a physical device unit for use as a communications link. It turns the device on-line, sets device characteristics, and ensures that the appropriate data terminal is ready. IO.TRM disconnects the device. If the device has a dial-up interface, it also hangs up the line.

9.3.2.4 IO.RNS - The IO.RNS QIO function is used to read a logical block of data, without stripping the sync characters which may precede the data. A similar function is IO.RLB, which is non-transparent, in that it causes sync characters preceding the data message to be stripped. IO.RLB is used at the start of a segmented data request, in which the block might have the following layout:



where: S is a sync character

H is a header character

CS is a validity check character

The programmer must strip sync characters from the beginning of a data block in this way. Stripping only at the beginning of a read allows a later character which happens to have the same binary value as a sync character to be read without stripping. IO.RLB is used to read a logical block with leading sync characters stripped; IO.RNS is used to read the block without stripping leading sync characters. Since the DA11-B is a parallel device and there are no sync characters, it treats the latter as if it were IO.RLB. Generally, IO.RLB should be used.

9.3.2.5 IO.RWD - Completion of the IO.RWD means that rewind has been initiated and the magtape controller is free. Additional operations may then be initiated. Requests for the same drive are queued by the driver until beginning-of-tape (BOT) is reached.

9.3.2.6 IO.SYN - This QIO function allows the programmer to specify the sync character to be recognized when an IO.RLB or IO.WLB function is performed. IO.SYN can be combined (ORed together) with IO.HDX to set the characteristics of the physical device unit.

9.3.2.7 IO.WNS - This QIO function causes a logical block to be written with no preceding sync characters. To ensure that the two systems involved in a communication are synchronized, two or more sync characters are transmitted by one system and received by the other

before any other message can be sent. IO.WLB is used to write a block of data, preceded by sync characters; IO.WNS is used to perform a block transfer without sending sync characters first. Since the DA11-B is a parallel device and there are no sync characters, it treats the latter as if it were IO.WLB. Generally, IO.WLB should be used.

9.4 STATUS RETURNS

The error and status conditions listed in Table 9-4 are returned by the communication drivers described in this chapter.

Table 9-4
Communication Status Returns

Code	Reason
IS.SUC	Successful completion The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading or writing.
IS.PND	I/O request pending The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with zeros.
IE.BCC	Block check error When the Cyclic Redundancy Check (CRC) option is present on the DQ11, a check character is appended to each message transmitted. The receiver of the messages recalculates the check character and compares it with the one transmitted. This error code is returned when the two check characters do not match, and represents a transmission error.
IE.DAO	Data overrun Due to UNIBUS traffic or a modem problem, the DQ11 controller was unable to maintain the data rate required to prevent data loss (i.e., the receipt of another byte before processing of a previous byte was completed).
IE.DNR	Device not ready The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation. This code is returned to indicate one of the following conditions: . The physical device unit could not be initialized (i.e., the circuit could not be completed).

Table 9-4 (Cont.)
Communication Status Returns

Code	Reason
IE.DNR (Cont.)	. The transmission of a character was not followed by an interrupt within the period of time selected as the device timeout period. This timeout occurs only when a transmission is in progress and the interrupt marking completion of a message does not occur. The appropriate response to this condition is to attempt to resynchronize the device by initializing and accepting the next request. A timeout does not occur on a read. If the receiving device is not ready, the transfer will not be initiated by the transmitting device. Once the transfer is initiated, however, it will complete either by satisfying the requested byte count or by timing out.
IE.IFC	Illegal function A function code was specified in an I/O request that is illegal for message-oriented communication devices.
IE.OFL	Device off-line The physical device unit associated with the LUN specified in the QIO directive was not on-line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.SPC	Illegal address space The buffer specified for a read or write request was partially or totally outside the address space of the issuing task. Alternately, a byte count of zero was specified.
IE.VER	Nonrecoverable error (DA11B only) The data transfer terminated before all of the data has been transmitted. The error code is returned on transmit when both systems attempt to transmit at the same time. This condition is detected by the device protocol. The error code is returned on receive when the transmit data count of the transmitting side does not equal the data count specified by the receive QIO.

9.5 PROGRAMMING HINTS

This section contains information on important programming considerations relevant to users of the message-oriented communication interfaces described in this chapter.

9.5.1 Transmission Validation

Because there is no way for the transmitting device to verify that the data block has successfully arrived at the receiving device unless the receiver responds, the transmitter assumes that any message which is clocked out on the line (without line or device outage) has been successfully transmitted. As soon as the receiver is able to satisfy a read request, it returns a successful status code (IS.SUC) in the I/O status block. Of course, only the task which receives the message can determine whether or not the message has actually been transmitted accurately.

The receiving device should be ready to receive data (with a read request) at the time the transmission is sent.

9.5.2 Redundancy Checking

By the nature of message-oriented communications, only the task which receives a communication can determine whether or not the message was received successfully. The transmitter simply transfers data, without validation of any kind. It is therefore the responsibility of the communicating tasks which use the device to check the accuracy of the transmission. A simple validity check is a checksum-type longitudinal redundancy check. A better approach to validating data is the use of a cyclic redundancy check (CRC). A CRC can be computed in software or with a hardware device, such as the KG-11 communications arithmetic option.

9.5.3 Half-Duplex and Full-Duplex Considerations

Because there is a single I/O request queue, only one QIO request can be performed at a time. It is therefore not possible, through QIOs, for a device to send and receive data at the same time. Also, since timeouts are not set for receive functions, a receive QIO is terminated only by receiving a message from the remote system, or by issuing an IO.KIL QIO for the device. Therefore, if no message is transmitted by the remote system, a receive will not terminate, and no further I/O can be performed on that device until the receive is killed by issuing an IO.KIL QIO.

Both half-duplex and full-duplex lines can be used with the DL11-E, DU11, DP11, and DQ11. The mode is settable by using IO.FDX for full-duplex and IO.HDX for half-duplex. In half-duplex mode, the modem signal RTS (Request To Send) is cleared after each "transmit message". In full-duplex, this signal is always left on. Using full-duplex mode eliminates modem delays in transmission, but requires full-duplex hardware and communication links.

Only half-duplex mode is available with the DA11-B because of the nature of the hardware.

9.5.4 Low-Traffic Sync Character Considerations

If message traffic on a line is low, each message sent from a communications device should be preceded by a sync train. This

enables the controller to resynchronize if a message is "broken" (i.e., part or all of it is lost in transmission). Correspondingly, every message received by a communications device under low-traffic conditions, when messages are not contiguous (back-to-back), should be read via an IO.RLB (read, strip sync) function. This requires that the first character in the data message itself not have the binary value of the sync character.

9.5.5 Vertical Parity Support

Vertical parity is not supported by the DA11-B, DL11-E, DP11, DQ11, or DU11. Codes are assumed to be eight-bit only.

9.5.6 Importance of IO.INL

After the type of communication line has been determined, and after IO.SYN has specified the sync character, it is extremely important that IO.INL be issued before any transfers occur. This ensures that appropriate parameters are initialized and that the interface is properly conditioned. Note that IO.INL provides the only means of setting device characteristics, such as sync character. For this reason, IO.INL should always be used immediately prior to the first transfer over a newly-activated link.

9.6 PROGRAMMING EXAMPLE

The following example illustrates the initialization, setting of device parameters, and transmission of a block of data on a message-oriented communication device.

```
        .MCALL  ALUN$S,QIO$S
        .       .
        .       .
        .       .
        ALUN$S  #1,#"XP,#0                          ; USE LUN1 FOR DP11
        QIO$S   #IO.HDX!IO.SYN,#1,,,,,<#226>        ; SET DEVICE PARAMETERS
        QIO$S   #IO.INL,#1                          ; PUT DEVICE ON LINE
        QIO$S   #IO.WLB,#1,,,#TXSTS,#TXAST,<#TXBUF,#100>; SEND A BLOCK
        .       .
        .       .
        .       .
TXAST:  CMPB    #IS.SUC&377,@(SP)+                  ; WAS DATA CLOCKED OUT
                                                    ; SUCCESSFULLY?
                                                    ; IF SO, SET UP FOR NEXT
        BEQ     10$                                 ; BLOCK
```

CHAPTER 10

ANALOG-TO-DIGITAL CONVERTER DRIVERS

10.1 INTRODUCTION

The AFC11 and AD01-D analog-to-digital (A/D) converters are used for the acquisition of industrial and laboratory analog data. Although each has its own driver, programming for both is quite similar and both are multichannel, programmable gain devices. The AD01-D should not be confused with the ADU01, a UDC module, which is described in Chapter 11. Table 10-1 compares the AFC11 and the AD01-D briefly, and subsequent sections describe these devices in greater detail.

Table 10-1
Standard Analog-to-Digital Converters

	AFC11	AD01-D
Maximum sampling rate (points per second)	200 (20 per single channel)	Approximately 10,000
Number of bits	13 or 14	10 or 11
Maximum number of analog channels that can be multiplexed	1024	64

10.1.1 AFC11 Analog-to-Digital Converter

The AFC11 is a differential analog input subsystem for industrial data-acquisition and control systems. It multiplexes signals, selects gain, and performs a 13- or 14-bit analog-to-digital conversion under program control. With the use of appropriate signal-conditioning modules, the system can intermix and accept low-level, high-level, and current inputs, with a high degree of noise immunity.

10.1.2 AD01-D Analog-to-Digital Converter

The AD01-D is an extremely fast analog data-acquisition system. It multiplexes signals, selects gain, and performs a 10- or 11-bit analog-to-digital conversion under program control. The AD01-D is normally unipolar, but an optional sign-bit facilitates bipolar operation.

10.2 GET LUN INFORMATION MACRO

If a GET LUN INFORMATION system directive is issued for a LUN associated with an analog-to-digital converter, word 2 (the first characteristics word) contains all zeros, words 3 and 4 are undefined, and word 5 is not significant, since there is no concept of a default buffer size for analog-to-digital converters.

10.3 QIO MACRO

This section summarizes standard and device-specific QIO functions for analog-to-digital converter drivers.

10.3.1 Standard QIO Function

The standard function that is valid for analog-to-digital converters is shown in Table 10-2.

Table 10-2
Standard QIO Function for the A/D Converters

Format	Function
QIO$C IO.KIL,...	Cancel I/O requests

Since all requests are processed within a small amount of time, no in-progress request is ever cancelled. This function simply cancels all queued requests.

10.3.2 Device-Specific QIO Function

The device-specific function of the QIO macro that is valid for analog-to-digital converters is shown in Table 10-3.

Table 10-3
Device-Specific QIO Function for the A/D Converters

Format	Function
QIO$C IO.RBC,...,<stadd,size,stcnta>	Initiate multiple A/D conversions

where: stadd is the starting address of the data buffer (must be on a word boundary).

size is the control buffer size in bytes (must be even and greater than zero); the data buffer is the same size.

stcnta is the starting address of the control buffer (must be on a word boundary); each control buffer word must be constructed as shown in Table 10-4.

Table 10-4
A/D Conversion Control Word

Bits	Meaning				AFC11	AD01-D
0-11	Channel number				Range: 0-1023	Range: 0-63
12-15	Gain value for this sample, expressed as a bit pattern as follows				Gain:	Gain:
	15	14	13	12		
	0	0	0	0	1	1
	0	0	0	1	2	2
	0	0	1	0	illegal	4
	0	0	1	1	illegal	8
	0	1	0	0	10	illegal
	0	1	0	1	20	illegal
	0	1	1	0	illegal	illegal
	0	1	1	1	illegal	illegal
	1	0	0	0	50	illegal
	1	0	0	1	100	illegal
	1	0	1	0	illegal	illegal
	1	0	1	1	illegal	illegal
	1	1	0	0	200	illegal
	1	1	0	1	1000	illegal
	1	1	1	0	illegal	illegal
	1	1	1	1	illegal	illegal

10.4 FORTRAN INTERFACE

A collection of FORTRAN-callable subroutines provide FORTRAN programs access to the AFC11 and the AD01-D. These are described in this section. All are reentrant and may be placed in a resident library.

10.4.1 Synchronous and Asynchronous Process Control I/O

The ISA standard provides for synchronous and asynchronous I/O. Synchronous I/O is indicated by appending a "W" to the name of the subroutine (e.g., AISQ/AISQW). The synchronous call suspends task execution until the I/O operation is complete. If the asynchronous form is used, execution continues and the calling program must periodically test the status word for completion.

10.4.2 The isb Status Array

The isb (I/O status block) parameter is a two-word integer array that contains the status of the FORTRAN call, in accordance with ISA convention. This array serves two purposes:

1. It is the 2-word I/O status block to which the driver returns a status code on completion of an I/O operation.

2. The first word of isb receives a status code from the FORTRAN interface in ISA-compatible format, with the exception of the I/O pending condition, which is indicated by a status of zero. The ISA standard code for this condition is +2.

The meaning of the contents of isb varies, depending on the FORTRAN call that has been executed, but Table 10-5 lists certain general principles that apply. The section describing each subroutine provides further details.

Table 10-5
Contents of First Word of isb

Contents	Meaning
isb(1) = 0	Operation pending; I/O in progress
isb(1) = 1	Successful completion
isb(1) = 3	Interface subroutine unable to generate QIO directive or number of samples is zero
3 < isb(1) $\leq$ 300	QIO directive rejected and actual error code = -(isb(1) - 3)
isb(1) > 300	Driver rejected request and actual error code = -(isb(1) - 300)

Unless otherwise specified, the value of isb(2) is the value returned by the driver to the second word of the I/O status block.

FORTRAN interface subroutines depend on asynchronous system traps to set their status. Thus, if the trap mechanism is disabled, proper status cannot be set.

10.4.3 FORTRAN Subroutine Summary

Table 10-6 lists the FORTRAN interface subroutines supported for the AFC11 and AD01-D under RSX-11M.

Table 10-6
FORTRAN Interface Subroutines for the AFC11 and AD01-D

Subroutine	Function
AIRD/AIRDW	Perform input of analog data in random sequence
AISQ/AISQW	Read a series of sequential analog input channels
ASADLN	Assign a LUN to the AD01-D
ASAFLN	Assign a LUN to the AFC11

The following subsections briefly describe the function and format of each FORTRAN subroutine call. Note the use of ASADLN and ASAFLN to assign a default logical unit number.

10.4.4 AIRD/AIRDW: Performing Input of Analog Data in Random Sequence

The ISA standard AIRD/AIRDW FORTRAN subroutines input analog data in random sequence. These calls are issued as follows:

```
CALL {AIRD } (inm,icont,idata,[isb],[lun])
     {AIRDW}
```

where: inm specifies the number of analog input channels.

icont is an integer array containing terminal connection data - channel number (right-justified in bits 0-11) and gain (bits 12-15), as shown in Table 10-4.

idata is an integer array to receive the converted values.

isb is a two-word integer array to which the subroutine status is returned

lun is the logical unit number.

The isb array has the standard meaning defined in section 10.4.2. If inm = 0, then isb(1) = 3. The contents of idata are undefined if an error occurs.

10.4.5 AISQ/AISQW: Reading Sequential Analog Input Channels

The ISA standard AISQ/AISQW FORTRAN subroutines read a series of sequential analog input channels. These calls are issued as follows:

```
CALL {AISQ } (inm,icont,idata,[isb],[lun])
     {AISQW}
```

where: inm specifies the number of analog input channels.

icont is an integer array containing terminal connection data - channel number (right-justified in bits 0-11) and gain (bits 12-15), as shown in Table 10-4.

idata is an integer array to receive the converted values.

isb is a 2-word integer array to which the subroutine status is returned.

lun is the logical unit number.

For sequential analog input, channel number is computed in steps of one, beginning with the value specified in the first element of icont. The channel number field is ignored in all other elements of the array.

The gain used for each conversion is taken from the respective element in icont. Thus, even though the channel number is ignored in all but the first element of icont, the gain must be specified for each conversion to be performed.

The isb array has the standard meaning defined in section 10.4.2. If inm = 0, then isb(1) = 3. The contents of idata are undefined if an error occurs.

10.4.6 ASADLN: Assigning a LUN to the AD01-D

The ASADLN FORTRAN subroutine assigns the specified LUN to the AD01-D and defines it as the default logical unit number to be used whenever a LUN specification is omitted from an AIRD(W)/AISQ(W) subroutine call. It is issued as follows:

```
CALL ASADLN (lun,[isw],[iun])
```

where:

- lun is the logical unit number to be assigned to the AD01-D and defined as the default unit.
- isw is an integer variable to which the result of the ASSIGN LUN system directive is returned.
- iun is the unit number to be assigned. If unspecified, a value of 0 is assumed.

Only the LUN specified in the last call to ASADLN or ASAFLN is defined as the default unit.

10.4.7 ASAFLN: Assigning a LUN to the AFC11

The ASAFLN FORTRAN subroutine assigns the specified LUN to the AFC11 and defines it as the default logical unit number to be used whenever a LUN specification is omitted from an AIRD(W)/AISQ(W) subroutine call. It is issued as follows:

```
CALL ASAFLN (lun,[isw],[iun])
```

where:

- lun is the logical unit number to be assigned to AFC11 and defined as the default unit.
- isw is an integer variable to which the status from the ASSIGN LUN system directive is returned.
- iun is the unit number to be assigned. If unspecified, a value of 0 is assumed.

Only the LUN specified in the last call to ASAFLN or ASADLN is defined as the default unit.

10.5 STATUS RETURNS

The error and status conditions listed in Table 10-7 are returned by the analog-to-digital converter drivers described in this chapter.

Table 10-7
A/D Converter Status Returns

Code	Reason
IS.SUC	Successful completion The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of A/D conversions performed.
IS.PND	I/O request pending The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with zeros.
IE.ABO	Operation aborted The specified I/O operation was cancelled via IO.KIL while still in the I/O queue.
IE.BAD	Bad parameter An illegal specification was supplied for one or more of the device-dependent QIO parameters (words 6-11). For the analog-to-digital converters, this code indicates that a bad channel number or gain code was specified in the control buffer.
IE.BYT	Byte-aligned buffer specified Byte alignment was specified for a data or control buffer, but only word alignment is legal for analog-to-digital converters. Alternately, the length of the data and control buffer is not an even number of bytes.
IE.DNR	Device not ready The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation. For the AFC11, this code is returned if an interrupt timeout occurred or the power failed. In the case of the AD01-D, which is not operated in interrupt mode, this code indicates a software timeout occurred (i.e. a conversion did not complete within 30 microseconds).
IE.IFC	Illegal function A function code was specified in an I/O request that is illegal for analog-to-digital converters.

Table 10-7 (Cont.)
A/D Converter Status Returns

Code	Reason
IE.OFL	Device off-line The physical device unit associated with the LUN specified in the QIO directive was not on-line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.SPC	Illegal address space The data or control buffer specified for a conversion request was partially or totally outside the address space of the issuing task. Alternately, a byte count of zero was specified.

FORTRAN interface values for these subroutines are presented in section 10.5.1.

10.5.1 FORTRAN Interface Values

The values listed in Table 10-8 are returned in FORTRAN subroutine calls.

Table 10-8
FORTRAN Interface Values

Status Return	FORTRAN Value
IS.SUC	+01
IS.PND	+00
IE.ABO	+315
IE.ADP	+101
IE.BAD	+301
IE.BYT	+319
IE.DAO	+313
IE.DNR	+303
IE.IEF	+100
IE.IFC	+302
IE.ILU	+99
IE.NOD	+323
IE.ONP	+305
IE.PRI	+316
IE.RSU	+317
IE.SDP	+102
IE.SPC	+306
IE.ULN	+08
IE.UPN	+04

10.6 FUNCTIONAL CAPABILITIES

The AFC11 and AD01-D operate only in multi-sample mode, because the user can simulate single-sample mode by simply specifying one sample. Multi-sample mode permits many channels to be sampled at approximately the same time without requiring the user to queue multiple I/O requests.

The maximum number of channels in the configuration is specified at system-generation time. This value is stored in the respective AFC11 and AD01-D unit control blocks.

10.6.1 Control and Data Buffers

The user must define two buffers of equal size, the control buffer and the data buffer. The former contains the control words needed to perform one A/D conversion per channel specified. Each control word indicates the channel to be sampled and the gain to be applied (see Table 10-4).

The data buffer receives the results of the conversions. Each result is placed in the data buffer location that corresponds to the control word that specified it.

10.7 PROGRAMMING HINTS

This section contains information on important programming considerations relevant to users of the analog-to-digital converter drivers described in this chapter.

10.7.1 Use of A/D Gain Ranges

Note that the A/D gain ranges overlap. The key to successful use of the A/D converters is to change to a higher gain whenever a full-scale reading is imminent and to change to a lower gain whenever the last A/D value recorded was less than half of full scale. This method maintains maximum resolution while avoiding saturation.

10.7.2 Identical Channel Numbers on the AFC11

When requesting sampling of more than one channel, the user should not specify multiple sampling of a single channel without 10 or more intervening samples on other channels. This ensures 50 milliseconds between samples on a single channel. If sampling occurs more often than this on a single channel, partial results are returned (see 10.7.3 below).

10.7.3 AFC11 Sampling Rate

Although the AFC11 can sample a maximum of 200 points per second, a single channel can only be sampled at 20 points per second. Because

the channel capacitor needs 50 milliseconds to recharge after each conversion, more frequent sampling may result in partial readings. If this occurs, the user will receive no indication that information is being lost. To ensure that information is not lost on any one channel, the user should sample approximately ten other channels before returning to the first one.

10.7.4 Restricting the Number of AD01-D Conversions

The AD01-D is an extremely fast device, providing a 25-microsecond conversion rate, and is driven programmably to minimize system overhead. However, an excessive number of conversions in a single request essentially locks out the rest of the system because the driver does not return control to the system until it has finished all the specified conversions. No other task can run, although interrupts can still occur and are processed.

CHAPTER 11

UNIVERSAL DIGITAL CONTROLLER DRIVER

11.1 INTRODUCTION

The UDC11 is a digital input/output system for industrial and process control applications. It interrogates and/or drives up to 252 directly addressable digital sense and/or control modules. The UDC11 operates under program control as a high-level digital multiplexer, interrogating digital inputs and driving digital outputs.

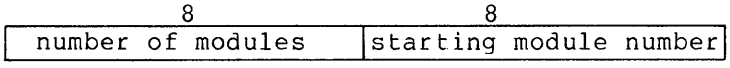
The UDC driver will support either the UDC11 or ICS11 subsystem. ICS11 (Industrial Control Subsystem) operates as an input/output device that is functionally similar to the UDC11. A maximum of 16 I/O modules can be placed in one ICS11 subsystem. Up to 12 ICS11s can be interfaced to one computer system. The ICS11 subsystem is also supported by the ICS/ICR-11 driver described in chapter 14. The reader should consult that chapter for a comparison of driver features.

While performing analog-to-digital conversions, the UDC11 driver can handle other functions, such as contact or timer interrupts or latching output. These functions are performed immediately, without requiring any in progress analog-to-digital conversions to first be completed.

Unlike other RSX-11M I/O device drivers, the UDC11 driver is neither a multicontroller nor a multiunit driver.

11.1.1 Creating the UDC11 Driver

Each installation must assemble the driver source module with a prefix file that defines the particular hardware configuration. The prefix file is created during system generation according to the user's response to questions relating to the UDC11. This file is named RSXMC.MAC and includes symbolic definitions of the UDC11 configuration. These definitions encode the relative module number and the number of modules for each generic type specified in the system generation dialog. The encoding has the following format:



One or more of the following symbols is generated:

Symbol	Module Type
U$$ADM	Analog input
U$$AOM	Analog output
U$$CIM	Contact interrupt
U$$CSM	Contact sense input
U$$LTM	Latching digital output
U$$SSM	Single-shot digital output
U$$TIM	Timer (I/O counter)

Note that all modules of a given type must be installed together in sequential slots.

11.1.2 Accessing UDC11 Modules

RSX-11M provides two methods of accessing the UDC11:

1. A QIO macro call issued to the driver
2. Restricted direct access by any task to I/O page registers dedicated to the UDC11

The first method, access through the driver, is required to service interrupting modules and to set and record the state of latching digital output modules.

The second method, direct access, is a high-speed, low-overhead way to service noninterrupting modules. The following functions may be performed in this manner:

- Analog output
- Contact sense input
- Single-shot digital output
- Read a contact interrupt module
- Read a timer module

11.1.2.1 Driver Services - The driver services the following types of modules:

1. Contact interrupt
2. Timer (I/O counter)
3. Analog input
4. Latching digital output

Contact and timer interrupts need not be serviced by a single task. One task may be connected to contact interrupts, and another to timer interrupts. A nonprivileged task can connect to either or both of

these classes by providing a circular buffer to receive interrupt information and an event flag to allow triggering of the task whenever a buffer entry is made.

11.1.2.2 Direct Access - A global common block within the I/O page provides restricted direct access to the UDC11 device registers. In a mapped system, the length of the block is set to prevent access to other device registers. In an unmapped system, the use of the common block is optional. Section 11.4 explains direct access more fully.

11.2 GET LUN INFORMATION MACRO

If a GET LUN INFORMATION system directive is issued for a LUN associated with the UDC11, word 2 (the first characteristics word) contains all zeros, words 3 and 4 are undefined, and word 5 is not significant, since there is no concept of a default buffer size for universal digital controllers.

11.3 QIO MACRO

This section summarizes standard and device-specific QIO functions for the UDC11 driver. In issuing them, note the numbering conventions described in 11.7.2.

11.3.1 Standard QIO Function

The standard function that is valid for the UDC11 is shown in Table 11-1.

Table 11-1
Standard QIO Function for the UDC11

Format	Function
QIO$C IO.KIL,...	cancel I/O requests

IO.KIL cancels all queued requests and disconnects all interrupt connections, but does not stop any I/O that is currently in progress.

11.3.2 Device-Specific QIO Functions

Table 11-2 summarizes device-specific QIO functions that are supported for the UDC11.

Table 11-2
Device-Specific QIO Functions for the UDC11

Format	Function
QIO$C IO.CCI,...,<stadd,sizb,tevf>	Connect a buffer to contact interrupts
QIO$C IO.CTI,...,<stadd,sizb,tevf,arv>	Connect a buffer to timer interrupts
QIO$C IO.DCI,...	Disconnect a buffer from contact interrupts
QIO$C IO.DTI,...	Disconnect a buffer from timer interrupts
QIO$C IO.ITI,...,<mn,ic>	Initialize a timer
QIO$C IO.MLO,...,<opn,pp,dp>	Open or close latching digital output points
QIO$C IO.RBC,...,<stadd,size,stcnta>	Initiate multiple A/D conversions

where: stadd is the starting address of the data buffer (must be on a word boundary).

sizb is the data buffer size in bytes (must be even and large enough to include a 2-word buffer header plus one data entry; the buffer may cross a 4K boundary).

tevf is the trigger event flag number (in range 1 through 64).

arv is the starting address of the table of initial/reset values (must be on a word boundary).

mn is the module number.

ic is the initial count.

opn is the first latching digital output point number, which must be on a module boundary (evenly divisible by 16).

pp is the 16-bit mask.

dp is the data pattern.

size is the control buffer size in bytes (must be even and greater than zero); the data buffer is the same size.

stcnta is the starting address of the control buffer (must be on a word boundary); each control buffer word must be constructed as shown in Table 11-3.

The following sections describe the functions listed in Table 11-2.

Table 11-3
A/D Conversion Control Word

Bits	Meaning	ADU01
0-11	Channel number	Range: 0-4095
12-15	Gain value for this sample, expressed as a bit pattern as follows	Gain:

15	14	13	12	
0	0	0	0	1
0	0	0	1	2
0	0	1	0	illegal
0	0	1	1	illegal
0	1	0	0	10
0	1	0	1	20
0	1	1	0	illegal
0	1	1	1	illegal
1	0	0	0	50
1	0	0	1	100
1	0	1	0	illegal
1	0	1	1	illegal
1	1	0	0	200
1	1	0	1	1000
1	1	1	0	illegal
1	1	1	1	illegal

11.3.2.1 Contact Interrupt Digital Input (W733 Modules) - Digital input and change of state information from contact interrupt modules is reported in a requester-provided circular buffer. The buffer consists of a 2-word header, followed by a data area in the following format:

1	driver index
2	user index
3	entry
4	entry
.	.
.	.
.	.

Whenever a change of state occurs in one or more contact points an interrupt is generated. The UDC11 driver gains control, determines whether the change of state is of interest (i.e., a contact closure and point closing (PCL) is set on the module), and then optionally makes an entry in the data area of the buffer, updates the index words and sets the trigger event flag of the connected task.

Each entry consists of five words in the following format:

Word	Contents
0	Entry existence indicator
1	Change-of-state (COS) indicator
2	Module data (current point values)
3	Module number (interrupting module)
4	Generic code (interrupting module)

The driver enters data in the location currently indicated by the driver index. This pointer can be considered as a FORTRAN index into the buffer, i.e., the first location of the buffer is associated with the index 1. The beginning of the data area is the location of the first entry (index 3). Entries are made in a circular fashion, starting at the beginning of the data area, filling in order of increasing memory address to the end of the data area, and then wrapping around from the end to the beginning of the data area.

It is expected that the connected task will maintain its own pointer (the user index) to the location in the buffer where it is next to retrieve contact interrupt data. When a task is triggered by the driver, it should process data in the buffer starting at the location indicated by its pointer and continuing in a circular fashion until the two pointers are equal or a zero entry existence indicator is encountered. Equality of pointers means that the connected task has retrieved all the contact interrupt information that the driver has entered into the buffer.

The entry existence indicator is set nonzero when a buffer entry is made. When a requester has removed or processed an entry, he must clear the existence indicator in order to free the buffer entry position.

If data input occurs in a burst sufficient to overrun the buffer, data is discarded and a count of data overruns is incremented. The nonzero entry existence indicator also serves as an overrun indicator. A positive value (+1) indicates no overruns between entries; a negative value is the two's complement of the number of times data have been discarded between entries.

The module number indicates a module on which a change of state in the direction of interest has been recognized for one or more discrete points. The direction of the change may be from 0 to 1 or 1 to 0, depending on the PCL (point closing) and POP (point opening) module jumpers. The change of state (COS) indicator specifies which point or points of the module have changed state.

The bit position of an on-bit in the COS indicator provides the low-order bits (3-0) of a point number and the module number provides the high order bits (15-4). The module data indicates the logical value (polarity) of each point in the module at the time of the interrupt.

Contact interrupt data can be reported to only one task. The functions IO.CCI and IO.DCI in Table 11-2 are provided to enable a task to connect and disconnect from contact interrupts. If the connection is successful, the second word of the I/O status block contains the number of words passed per interrupt in the low-order byte and the initial FORTRAN index to the beginning of the data area in the high-order byte.

NOTE

The size of the data area must be a multiple of the entry size.

11.3.2.2 Timer (W734 I/O Counter Modules) - A timer (I/O counter) module is a clock that is initialized (loaded), counts up or down, and then causes an interrupt. The UDC11 driver treats such modules in a way similar to that in which it handles contact interrupts. The requester provides a circular buffer similar to that for contact interrupts. Each entry consists of four words in the following format:

Word	Contents
0	Entry existence indicator
1	Module data (current value)
2	Module number (interrupting module)
3	Generic code (interrupting module)

The IO.CTI function in Table 11-2 enables a task to connect to timer interrupts. The table of initial/reset values is used to initially load the timers and to reload them on interrupt (overflow). The table contains one word for each timer module. The contents of the first word are used to load the first module, and so forth. If a timer has a nonzero value when it interrupts, it is not reloaded, so that self-clocking modules and modules that interrupt on half count can continue counting from the initial value.

The IO.DTI function in Table 11-2 disconnects a task from timer interrupts, and the IO.ITI function provides the capability of initializing a single timer. Requests to initialize a counter are valid only if the issuing task has connected a buffer for receiving counter interrupts.

NOTE

The size of the data area must be a multiple of the entry size.

11.3.2.3 Latching Digital Output (M685, M803, and M805 Modules) - Each module has 16 latching digital output points. The IO.MLO function in Table 11-2 opens or closes a set of up to 16 points. Bit n of the mask and data pattern corresponds to the point opn + n. If a bit in the mask is set, the corresponding point is opened or closed, depending on whether the corresponding bit in the data pattern is clear or set. If a bit in the mask is clear, the corresponding point remains unaltered.

11.3.2.4 Analog-to-Digital Converter (ADU01 Module) - Each ADU01 module has eight analog input channels. The IO.R8C function in Table 11-2 initiates A/D conversions on multiple ADU01 input channels.

Restrictions on maximum sampling rates are the same as defined for the AFC11 in Chapter 10.

11.3.2.5 ICS11 Analog-to-Digital Converter (IAD-IA module) - Each IAD-IA module has eight analog input channels. The channel capacity may be expanded to 120 by the addition of IMX-IA multiplexers. Each multiplexer adds 16 input channels to the converter. Restrictions on maximum sampling rates are the same as defined for the AFC11 in Chapter 10. The IAD-IA module preempts eight module slots regardless of the number of IMX-IA multiplexers installed.

For addressing purposes, each converter occupies a block of 120 channels. Thus, A/D converter 0 is addressed by referencing channels 0 through 119; A/D converter 1 is addressed by referencing channels 120 through 239 etc. When fewer than seven multiplexers are installed, not all addresses within the block are valid.

11.4 DIRECT ACCESS

Section 11.1.2 describes UDC11 functions that may be performed by referencing a module through its physical address in the I/O page. Under RSX-11M such access is accomplished by one of the following methods:

1. A privileged task or any task running in an unmapped system has unrestricted access to the I/O page and may therefore access each module by absolute address.

2. Using the Task Builder, a task may link to a global common area whose physical address limits span a set of locations in the I/O page. This method applies to either a mapped or unmapped system.

The latter method allows a task to be transported to any other system simply by relinking. Further, in a mapped system the memory management hardware will abort all references to device registers outside the physical address limits of the common block.

The operations required to implement each method may be summarized as follows:

1. Unrestricted access to the I/O page

   a. An object module is created which defines the UDC11 configuration through a list of absolute global addresses and addressing limits for each module type.

   b. The object module is included in the system library file.

   c. A task is created containing the appropriate global references. Such references are resolved when the task builder automatically searches the system library file.

Steps a and b are executed once, during system generation (see RSX-11M System Generation Manual). Step c is performed each time a task is created that references the UDC11.

2. Access to the I/O page through a Global Common Block:

   a. An object module is created which defines the UDC11 configuration through a list of relocatable global addresses and addressing limits for each module type.

   b. The object module is linked, using the Task Builder, to create an image of the Global Common block on disk.

   c. The SET command is used to define a common block that resides on the I/O page.

   d. The INSTALL MCR command is used to make the Global Common Block resident in memory.

   e. A task is created containing the appropriate global references. Such references are resolved by directing the Task Builder to link the Task to the common block.

Steps a through d are executed once, during system generation. Step e is performed each time a task is created that references the UDC11 common block. The following paragraphs describe each step in detail.

11.4.1 Defining the UDC11 Configuration

The source module UDCOM.MAC*, when assembled with the proper prefix file, provides global definitions for the following parameters:

- The starting address of each module type.
- The highest point number within a given module type.
- The highest module number within a given module type.

The last two parameters are absolute quantities that may be used to prevent a task from referencing a module that is nonexistent or out of limits.

By means of conditional assembly the list of addresses may be created as absolute symbols defining locations in the I/O page or as symbols within a relocatable program section to be used when building and linking to the UDC11 Global Common area.

11.4.1.1 Assembly Procedure for UDCOM.MAC

- UDCOM.MAC is assembled with the RSX-11M configuration parameters contained in the file RSXMC.MAC.

To create relocatable module addresses either the parameter U$$DCM or M$$MGE must be defined. M$$MGE will be included in RSXMC.MAC if memory management was specified when the system was generated. If not, the user should edit the file to include the following definition:

```
U$$DCM=0
```

* This module resides on the RK05 cartridge of the RSX-11M RK distribution kit labeled EXECUTIVE SOURCE. For RP distribution kits, it resides on the RP image. The file is located under UIC [11,10].

The file may then be assembled using the MCR command:

```
>MAC UDCOM,UDLST=[11,10]RSXMC,UDCOM
```

This command invokes the MACRO-11 assembler which processes the input files RSXMC.MAC and UDCOM.MAC to create UDCOM.OBJ and UDLST.LST.

To create absolute module addresses, both of the above parameters must be undefined. Edit RSXMC.MAC, if necessary, to remove definitions and then invoke the MACRO-11 assembler with the following MCR command:

```
>MAC UDCDF,UDLST=[11,10]RSXMC,UDCOM
```

In this sequence the files UDCDF.OBJ and UDLST.LST are created from the specified source modules. UDCDF.OBJ contains the module addresses in absolute form.

11.4.1.2 Symbols Defined by UDCOM.MAC - This section lists the symbolic definitions created by UDCOM.MAC.

The following symbols define the absolute or relocatable address of the first module of a given type:

Symbol	Module Type
$.ADM	Analog input
$.AOM	Analog output
$.CIM	Contact interrupt
$.CSM	Contact sense input
$.LTM	Latching digital output
$.SSM	Single-shot digital output
$.TIM	Timer (I/O counter)

The addresses in relocatable form are defined in a program section named UDCOM having the attributes:

```
REL - relocatable
OVR - overlaid
D   - data
GBL - global scope
```

Note that these attributes correspond to those attached to a named common block within a Fortran program.

In either the absolute or relocatable case, individual modules are referenced by the corresponding symbolic address plus a relative module index.

The following symbols define the highest digital point within a module type:

Symbol	Module Type
P$.CIM	Contact interrupt
P$.CSM	Contact sense input
P$.LTM	Latching digital output
P$.SSM	Single-shot digital output

The highest point number is defined relative to the first point on the first module of a specific type.

For example if two contact interrupt modules are installed, the symbol P$.CIM will have an octal value of 37.

The following symbols define the highest module number within a given module type.

Symbol	Module Type
M$.ADM	Analog input
M$.AOM	Analog output
M$.CIM	Contact interrupt
M$.CSM	Contact sense input
M$.LTM	Latching digital output
M$.SSM	Single-shot digital output
M$.TIM	Timer (I/O counter)

The highest module number is defined relative to the first module of a given type. Thus, based on the previous example, M$.CIM will have a value of 1.

11.4.2 Including UDC11 Symbolic Definitions in the System Object Module Library

As described in 11.4, a task having unrestricted access to the I/O page may reference a UDC11 module by absolute address. The object module UDCDF contains symbolic definitions of absolute module addresses and may be included in the System Object Module Library:

```
SY:[1,1]SYSLIB.OLB
```

The Task Builder automatically searches this file to resolve any undefined globals remaining after all input files have been processed.

The following example illustrates the procedure for including the file UDCDF.OBJ in the library.

```
>SET /UIC=[1,1]
>LBR SYSLIB/IN=[200,200]UDCDF
```

The SET MCR command is issued to establish the current UIC as [1,1]. Next, the RSX11M Librarian is invoked and instructed, through the use of the /IN switch to include the object module UDCDF.OBJ in the file SYSLIB.OLB.

11.4.3 Referencing the UDC11 through a Global Common Block

The following sections define the procedure for creating a Global Common block in the I/O PAGE, making the block resident in memory, and creating a task which references UDC11 modules within the block. Examples are given for both mapped and unmapped systems.

11.4.3.1 Creating a Global Common Block - The following sequence illustrates the use of the object file UDCOM.OBJ to create a disk image of the global common area in a mapped system.

```
>SET /UIC=[1,1]
>TKB
TKB>UDCOM/MM,LP:,SY:UDCOM/PI/-HD=[200,200]UDCOM
TKB>/
ENTER OPTIONS:
TKB>PAR=UDCOM:0:1000
TKB>STACK=0
TKB>/
```

In the above example, a current UIC of [1,1] is established and the Task Builder is initiated. The initial input line to the Task Builder specifies the following files:

- A core image output file to be named UDCOM.TSK
- A memory map output to the line printer
- A symbol table file to be named UDCOM.STB

All files reside on SY: under UIC [1,1]. The single input file, UDCOM.OBJ containing the UDC11 address definitions as relocatable values, constitutes the input.

The switches specified for the output files convey the following information to the Task Builder:

/MM	indicates that the core image of the common block will reside on a system with Memory Management.
/PI	indicates that the core image is position independent; that is the virtual address of the common block may appear on any 4K boundary within a task's address space.
/-HD	indicates that the core image will not contain a header. A header is only required for a core image file that is to be installed and executed as a task.

A single line of option input must be entered to eliminate the default memory allocation for the stack area.

The following sequence illustrates the corresponding procedure for an unmapped system:

```
>SET /UIC=[1,1]
>TKB
TKB>UDCOM/-MM,LP:,SY:UDCOM/PI/-HD=[200,200]UDCOM
TKB>/
ENTER OPTIONS:
TKB>STACK=0
TKB>PAR=UDCOM:171000:1000
TKB>/
```

Again the task builder is requested to produce a core image and symbol table file under the UIC [1,1] and a map file on the line printer from the input file UDCOM.OBJ. The output file switches convey the following information:

/-MM indicates that the core image of the common block will reside on an unmapped system.

/PI Indicates that the core image is position independent. In an unmapped system the core image is fixed in the same address space for all tasks; however, the global symbols defined in the symbol table file retain the relocatable attribute.

/-HD indicates that a core image without a header is to be created.

The PAR option specifies the base and length of the common area to coincide with the standard UDC11 addresses in the I/O page. All references to the common block by tasks will be resolved within this region.

11.4.3.2 Making the Common Block Resident - The following SET command creates a UDC11 common block residing in the I/O page for a mapped system:

```
>SET /MAIN=UDCOM:7710:10:DEV
```

The corresponding command in an unmapped system is:

```
>SET /MAIN=UDCOM:1710:10:DEV
```

The preceding sequence specifies the allocation of a common block in the I/O page whose physical address limits correspond to the UDC11 standard locations. Note that the address bounds and length are defined in units of 32 words.

The command

```
>INS [1,1]UDCOM
```

declares the common block resident in the system.

11.4.3.3 Linking a Task to the UDC11 Common Block - A task may access UDC11 modules by linking to the common block as follows:

```
TKB>TASK,LP:=TASK.OBJ
TKB>/
ENTER OPTIONS:
TKB> COMMON=UDCOM:RW
TKB>/
```

The above sequence is valid for either a mapped or unmapped system. In both cases the Task Builder will link the task to the common block by relocating the Global symbol definitions contained in UDCOM.STB. If memory management is present, the Executive will map the appropriate physical locations into the task's virtual addressing space when the task is made active.

11.5 FORTRAN INTERFACE

A collection of FORTRAN-callable subroutines provide FORTRAN programs access to the UDC11. These are described in this section. All are reentrant and may be placed in a resident library.

Instead of using the FORTRAN-callable subroutines described in this section, a FORTRAN program may use the global common feature described in section 11.4 to reference UDC11 modules directly in the I/O page, as shown in the following example:

```
C
C       UDC11 GLOBAL COMMON
C
        COMMON /UDCOM/ ICSM(10),IAO(10)
C
C       READ CONTACT SENSE MODULE 1 DIRECTLY
C
        ICS=ICSM(1)
```

Note that the position of each module type must correspond to the sequence in which storage is allocated in the common statements.

11.5.1 Synchronous and Asynchronous Process Control I/O

The ISA standard provides for synchronous and asynchronous process I/O. Synchronous I/O is indicated by appending a "W" to the name of the subroutine (e.g., AO/AOW). But due to the fact that nearly all UDC11 I/O operations are performed immediately, in most cases the "W" form of the call is retained only for compatibility and has no meaning under RSX-11M. In the case of A/D input, however, the "W" from is significant: the synchronous call suspends task execution until input is complete. If the asynchronous form is used, execution continues and the calling program must periodically test the status word for completion.

11.5.2 The isb Status Array

The isb (I/C status block) parameter is a 2-word integer array that contains the status of the FORTRAN call, in accordance with ISA (Instrument Society of America) convention. This array serves two purposes:

1. It is the 2-word I/O status block to which the driver returns an I/O status code on completion of an I/O operation.

2. The first word of isb receives a status code from the FORTRAN interface in ISA-compatible format, with the exception of the I/O pending condition, which is indicated by a status of zero. The ISA standard code for this condition is +2.

The meaning of the contents of isb varies, depending on the FORTRAN call that has been executed, but Table 11-4 lists certain general principles that apply. The section describing each subroutine gives more details.

Table 11-4
Contents of First Word of isb

Contents	Meaning
isb(1) = 0	Operation pending; I/O in progress
isb(1) = 1	Successful completion
isb(1) = 3	Interface subroutine unable to generate QIO directive or number of points requested is zero
3 < isb(1) $\leq$ 300	QIO directive rejected and actual error code = -(isb(1) - 3)
isb(1) > 300	Driver rejected request and actual error code = -(isb(1) - 300)

In some cases, the values or states of points being read, pulsed, or latched are returned to isb word 2.

FORTRAN interface subroutines for analog input depend on asynchronous system traps to set their status. Thus, if the trap mechanism is disabled, proper status cannot be set.

For direct access calls (indicated in Table 11-5 below), errors are detected and returned by the FORTRAN interface subroutine itself, rather than by the driver. Although the use of a two-word status block is therefore unnecessary, these errors are returned in standard format to retain compatibility with functions called through QIO directives and handled by other drivers. Errors of this type that may be returned are:

isb(1) = 3	Number of points requested is zero
isb(1) = +321	Invalid UDC11 module

11.5.3 FORTRAN Subroutine Summary

Table 11-5 lists the FORTRAN interface subroutines supported for the UDC11 under RSX-11M. (D) indicates a direct access call and the optional logical unit number for such a call may be specified to retain compatibility with RSX-11D, but this specification is ignored.

Table 11-5
FORTRAN Interface Subroutines for the UDC11

Subroutine	Function
AIRD/AIRDW	Perform input of analog data in random sequence
AISQ/AISQW	Read a series of sequential analog input channels

Table 11-5 (Cont.)
FORTRAN Interface Subroutines for the UDC11

Subroutine	Function
AO/AOW	Perform analog output on several channels (D)
ASUDLN	Assign a LUN to the UDC11
CTDI	Connect a circular buffer to receive contact interrupt data
CTTI	Connect a circular buffer to receive timer interrupt data
DFDI	Disconnect a buffer from contact interrupts
DFTI	Disconnect a buffer from timer interrupts
DI/DIW	Read several 16-point contact sense fields (D)
DOL/DOLW	Latch or unlatch several 16-point fields
DOM/DOMW	Pulse several 16-point fields (D)
RCIPT	Read the state of a single contact interrupt point (D)
RDCS	Read the contents of a contact interrupt circular buffer, returning data on only those points that have changed state.
RDDI	Read the contents of a contact interrupt circular buffer, one point for each call
RDTI	Read the contents of a timer interrupt circular buffer, one entry for each call
RDWD	Read the contents of a contact interrupt circular buffer, returning 16 bits of module data and change-of-state information.
RSTI	Read a single timer module (D)
SCTI	Set a timer module to an initial value

The following subsections briefly describe the function and format of each FORTRAN subroutine call. Note the use of ASUDLN to specify a default logical unit number. Also consider the numbering conventions described in 11.7.2.

The following FORTRAN functions do not perform I/O directly, but facilitate conversions between BCD and binary.

Convert four BCD digits to a binary number:

```
IBIN = KBCD2B(IBCD)
```

Convert a binary number to four BCD digits:

```
IBCD = KB2BCD(IBIN)
```

11.5.4 AIRD/AIRDW: Performing Input of Analog Data in Random Sequence

The ISA standard AIRD/AIRDW FORTRAN subroutines input analog data in random sequence. These calls are issued as follows:

```
CALL {AIRD } (inm,icont,idata,[isb],lun)
     {AIRDW}
```

where: inm specifies the number of analog input channels.

icont is an integer array containing terminal connection data - channel number (right-justified in bits 0-11) and gain (bits 12-15), as shown in Table 11-3.

idata is an integer array to receive the converted values.

isb is a two-word integer array to which the subroutine status is returned.

lun is the logical unit number.

NOTE

lun is a required parameter.

The isb array has the standard meaning defined in section 11.5.2. If inm = 0, then isb(1) = 3. The contents of idata are undefined if an error occurs.

11.5.5 AISQ/AISQW: Reading Sequential Analog Input Channels

The ISA standard AISQ/AISQW FORTRAN subroutines read a series of sequential analog input channels. These calls are issued as follows:

```
CALL {AISQ } (inm,icont,idata,[isb],lun)
     {AISQW}
```

where: inm specifies the number of analog input channels.

icont is an integer array containing terminal connection data - channel number (right-justified in bits 0-11) and gain (bits 12-15), as shown in Table 11-3.

idata is an integer array to receive the converted values.

isb is a two-word integer array to which the subroutine status is returned.

lun is the logical unit number.

NOTE

lun is a required parameter.

For sequential analog input, channel number is computed in steps of one, beginning with the value specified in the first element of icont. The channel number field is ignored in all other elements of the array.

The gain used for each conversion is taken from the respective element in icont. Thus, even though the channel number is ignored in all but the first element of icont, the gain must be specified for each conversion to be performed.

The isb array has the standard meaning defined in section 11.5.2. If inm = 0, then isb(1) = 3. The contents of idata are undefined if an error occurs.

11.5.6 AO/AOW: Performing Analog Output

The ISA standard AO/AOW FORTRAN subroutines initiate analog output on several channels. These calls are issued as follows:

CALL {AO / AOW} (inm,icont,idata,[isb],[lun])

where: inm specifies the number of analog output channels.

icont is an integer array containing the channel numbers.

idata is an integer array containing the output voltage settings, in the range 0-1023.

isb is a two-word integer array to which the subroutine status is returned.

lun is the logical unit number (ignored if present).

The isb array has the standard meaning defined in section 11.5.2.

11.5.7 ASUDLN: Assigning a LUN to the UDC11

The ASUDLN FORTRAN subroutine assigns the specified LUN to the specified unit and defines it as the default logical unit number to be used whenever a LUN specification is omitted from a UDC11 subroutine call. It is issued as follows:

```
CALL ASUDLN (lun,[isw],[iun])
```

where: lun is the logical unit number to be assigned to the specified unit, and defined as the default.

isw is an integer variable to which the result of the ASSIGN LUN system directive is returned.

iun is an integer defining the UDC11 unit number. If no number is specified, 0 is assumed.

11.5.8 CTDI: Connecting to Contact Interrupts

The CTDI FORTRAN subroutine connects a task to contact interrupts and specifies a circular buffer to receive contact interrupt data. The length of this buffer can be computed by considering the following:

- Rate at which contact module interrupts occur
- Number of modules that can interrupt simultaneously
- Rate at which the circular buffer is emptied

The UDC11 driver generates a five-word entry for each contact interrupt and the interface subroutine itself requires 10 words of additional storage. Thus the isz parameter, described below, can be computed as follows:

```
isz = (10 + 5 * n)
```

where n is the number of entries in the buffer and isz is expressed in words.

The call is issued as follows:

```
CALL CTDI (ibuf,isz,iev,[isb],[lun])
```

where: ibuf is an integer array that is to receive contact interrupt data.

isz is the length of the array in words, with a minimum size of 15.

iev is the trigger event flag number. The specified event flag is set whenever the driver inserts an entry in the data buffer.

isb is a 2-word integer array to which the subroutine status is returned.

lun is the logical unit number.

The isb array has the standard meaning defined in section 11.5.2.

11.5.9 CTTI: Connecting to Timer Interrupts

The CTTI FORTRAN subroutine connects a task to timer interrupts and specifies a circular buffer to receive timer interrupt data. The length of this buffer can be computed by considering the following:

- Rate at which timer module interrupts occur

- Number of modules that can interrupt simultaneously
- Rate at which the circular buffer is emptied

The UDC11 driver generates a four-word entry for each timer interrupt and the interface subroutine itself requires 8 words of additional storage. Thus the isz parameter, described below, can be computed as follows:

```
isz = (8 + 4 * n)
```

where n is the number of entries in the buffer and isz is expressed in words.

When a timer module interrupt occurs, the driver resets the count to an initial value, normally that specified in iv. The initial value for a specific module can be modified by calling the SCTI subroutine (see section 11.5.19).

The call is issued as follows:

```
CALL CTTI (ibuf,isz,iev,iv,[isb],[lun])
```

where:
- ibuf is an integer array that is to receive timer interrupt data.
- isz is the length of the array in words, with a minimum size of 12.
- iev is a trigger event flag number. The specified event flag is set whenever the driver inserts an entry in the data buffer.
- iv is an integer array which contains the initial timer module values, with one entry for each timer module, where entry n corresponds to timer module number n-1.
- isb is a 2-word integer array to which the subroutine status is returned.
- lun is the logical unit number.

The isb array has the standard meaning defined in section 11.5.2.

11.5.10 DFDI: Disconnecting from Contact Interrupts

The DFDI FORTRAN subroutine disconnects a task from contact interrupts. It is issued as follows:

```
CALL DFDI ([isb],[lun])
```

where:
- isb is a 2-word integer array to which the subroutine status is returned.
- lun is the logical unit number.

The isb array has the standard meaning defined in section 11.5.2.

11.5.11 DFTI: Disconnecting from Timer Interrupts

The DFTI FORTRAN subroutine disconnects a task from timer interrupts. It is issued as follows:

```
CALL DFTI ([isb],[lun])
```

where: isb is a 2-word integer array to which the subroutine status is returned.

lun is the logical unit number.

The isb array has the standard meaning defined in section 11.5.2

11.5.12 DI/DIW: Reading Several Contact Sense Fields

The ISA standard DI/DIW FORTRAN subroutines read several 16-point contact sense fields. These calls are issued as follows:

```
CALL {DI  } (inm,icont,idata,isb,[lun])
     {DIW }
```

where: inm specifies the number of fields to be read.

icont is an integer array containing the initial point number of each field to be read.

idata is an integer array that is to receive the input data, 16 bits of contact data for each field read.

isb is a 2-word integer array to which the subroutine status is returned.

lun is the logical unit number (ignored if present).

The isb array has the standard meaning defined in section 11.5.2.

11.5.13 DOL/DOLW: Latching or Unlatching Several Fields

The ISA standard DOL/DOLW FORTRAN subroutines latch or unlatch one or more 16-point fields. These calls are issued as follows:

```
CALL {DOL  } (inm,icont,idata,imsk,[isb],[lun])
     {DOLW }
```

where: inm specifies the number of fields to be latched or unlatched.

icont is an integer array containing the initial point number of each 16-point field.

idata is an integer array which specifies the points to be latched or unlatched; bit n of idata corresponds to point number icont + n; if the corresponding bit in imsk is set, the bit is changed; a bit value of 1 indicates latching, and 0 unlatching; each entry in the array specifies a string of 16 points.

imsk is an integer array in which bits are set to indicate points whose states are to be changed in the corresponding idata bits; each entry in the array specifies a 16-bit mask word.

isb is a 2-word integer array to which the subroutine status is returned.

lun is the logical unit number.

The isb array has the standard meaning defined in section 11.5.2.

11.5.14 DOM/DOMW: Pulsing Several Fields

The ISA standard DOM/DOMW FORTRAN subroutines pulse several 16-bit fields (one-shot digital output points). These calls are issued as follows:

CALL {DOM / DOMW} (inm,icont,idata,[idx],[isb],[lun])

where: inm specifies the number of fields to be pulsed.

icont is an integer array containing the initial point number of each 16-point field.

idata is an integer array which specifies the points to be pulsed; bit n of idata corresponds to point number icont + n.

idx is a dummy argument retained for compatibility with existing Instrument Society of America standard FORTRAN process control calls.

isb is a 2-word integer array to which the subroutine status is returned.

lun is the logical unit number (ignored if present).

The isb array has the standard meaning defined in section 11.5.2.

11.5.15 RCIPT: Reading a Contact Interrupt Point

The RCIPT FORTRAN subroutine reads the state of a single contact interrupt point. It is issued as follows:

```
CALL RCIPT (ipt,isb,[lun])
```

where: ipt is the number of the point to be read; points are numbered sequentially from 0, the first point on the first contact interrupt module.

isb is a 2-word integer array to which the subroutine status is returned.

lun is the logical unit number (ignored if present).

The isb array has the same basic meaning defined in section 11.5.2. However, isb word 2 is set to one of the following values, representing the state of the point:

Setting	Meaning
.FALSE. (0)	Point is open
.TRUE. (-1)	Point is closed

11.5.16 RDCS: Reading Contact Interrupt Change-of-State Data from a Circular Buffer

The RDCS FORTRAN subroutine reads contact interrupt data from a circular buffer that was specified in a CTDI call (see 11.5.8 above). It does no actual input or output, but rather performs a point-by-point scan of an interrupt entry in the buffer, returning the state of each point that has changed state, as a logical value. The trigger event flag which was specified in the CTDI call is cleared when the "buffer empty" condition is detected.

On the initial call to RDCS, the module number, module data, and change-of-state word of the next interrupt entry are read from the circular buffer and stored for subsequent reference. The subroutine then searches the entry change-of-state word until a nonzero point is encountered. The point number is computed and returned to the caller along with the state of the point. Scanning for points that have changed state resumes on the next call; all other points are bypassed. The next entry is automatically read when the caller has received all change-of-state information from the current entry. If a valid entry is not found, ipt is set negative and ict (if specified) is either assigned a value of zero or an overrun count maintained by the UDC11 driver. If ict is zero, no further entries remain. A nonzero value indicates that the driver received more data than could be stored in the buffer, and ict represents the number of entries that were discarded.

The RDCS call is issued as follows:

```
CALL RDCS (ipt,ival,[ict])
```

where: ipt is a variable to which the digital input point number is returned; it may be set as follows:

- ipt < 0 if no valid entry is found (i.e., no interrupt data currently in buffer, or overrun detected). One of the following values is returned to indicate the condition detected:

-1 = Buffer empty
-2 = Overrun detected

. ipt => 0 if the value indicated is a point number that has changed state; the state is returned to ival.

ival is a variable to which the state of the point is returned; it may be set as follows:

. .FALSE. (0) if the point is open

. .TRUE. (-1) if the point is closed

ict is an integer variable for receiving the overrun count. A nonzero positive count indicates that the driver was unable to store the number of interrupts indicated.

11.5.17 RDDI: Reading Contact Interrupt Data From a Circular Buffer

The RDDI FORTRAN subroutine reads contact interrupt data from a circular buffer that was specified in a CTDI call (see 11.5.8 above). It does no actual input or output, but rather performs a point-by-point scan of an interrupt entry in the buffer, returning the state of each point as a logical value. The trigger event flag which was specified in the CTDI call is also cleared.

On the initial call to RDDI the module number and data of the next interrupt entry are read from the circular buffer and stored for subsequent reference. The subroutine then sets the current data bit number n to zero, examines the state of data bit n, and converts bit n to a point number via the following formula:

```
ipt = module number * 16 + n
```

On each subsequent call, n is incremented by one and then data bit n is examined in the stored module data. When n reaches 16, it is reset to zero and an attempt is made to read the next interrupt entry from the circular buffer. If a valid entry is not found, ipt is set negative and ict (if specified) is either assigned a value of zero or an overrun count maintained by the UDC11 driver. If ict is zero, no further entries remain. A nonzero value indicates that the driver received more data than could be stored in the buffer, and ict represents the number of entries that were discarded.

The RDDI call is issued as follows:

```
CALL RDDI (ipt,ival,[ict])
```

where: ipt is a variable to which the digital input point number is returned; it may be set as follows:

. ipt < 0 if no valid entry is found (i.e., no interrupt data currently in buffer, or buffer empty). One of the following values is returned to indicate the condition detected:

-1=Buffer empty
-2=Overrun detected

- ipt => 0 if the value indicated is a point number; the state is returned to ival

ival is a variable to which the state of the point is returned; it may be set as follows:

- .FALSE. (0) if the point is open
- .TRUE. (-1) if the point is closed

ict is a variable to which the overrun count may be returned; a nonzero positive count indicates that the driver was unable to store the number of entries indicated.

11.5.18 RDTI: Reading Timer Interrupt Data From a Circular Buffer

The RDTI FORTRAN subroutine reads timer interrupt data from a circular buffer that was specified in a CTTI call (see 11.5.9 above). It does no actual input or output, but rather performs a scan of each entry in the buffer, returning the timer value for each call. The trigger event flag which was specified in the CTTI call is also cleared.

When a timer module interrupt occurs, the UDC11 driver resets the count to an initial value, usually that specified in the iv array on the CTTI call. The initial value can be modified for a specific module by calling the subroutine SCTI (see section 11.5.19).

The RDTI call is issued as follows:

```
CALL RDTI (imod,itm,[ivrn])
```

where: imod is a variable to which the module number is returned; it may be set as follows:

- imod < 0 if no valid entry is found (i.e., no interrupt data currently in buffer, or buffer empty). One of the following values is returned to indicate the condition detected:

  -1=Buffer empty
  -2=Overrun detected

- imod > 0 if the entry is valid, indicating a module number; the value of the timer module is returned in itm

itm is a variable to which the timer value is returned.

ivrn is a variable to which the overrun count may be returned; a nonzero positive count indicates that the driver was unable to store the number of values indicated.

11.5.19 RDWD: Reading a Full Word of Contact Interrupt Data from the Circular Buffer

The RDWD FORTRAN subroutine reads a full word of contact interrupt and change-of-state data from the circular buffer that was specified in a CTDI call (see section 11.5.8). It does no actual input or output, but rather performs a scan of each entry, returning the state of a module and optionally, the change-of-state data for each call. The trigger event flag specified in the call to CTDI is cleared.

The call to RDWD is issued as follows:

```
CALL RDWD (imod,ist,[ivrn],[icos])
```

where: imod is a variable to which the module number is returned; it may be set as follows:

- imod < 0 if no valid entry is found (i.e., no interrupt data currently in buffer or overrun detected). One of the following values is returned to indicate the condition detected:

  -1=Buffer empty
  -2=Overrun detected

ist is a variable to which the module data is returned.

ivrn is a variable to which the overrun count may be returned; a nonzero, positive count indicates that the driver was unable to store the number of entries indicated.

icos is a variable to which the change-of-state data is returned. One bit is set for each point that has changed state in the direction indicated by the "point open" (POP) or "point closed" (PCL) jumpers on the module.

11.5.20 RSTI: Reading a Timer Module

The RSTI FORTRAN subroutine reads a single timer module. It is issued as follows:

```
CALL RSTI (imod,isb,[lun])
```

where: imod is the module number of the timer to be read.

isb is a 2-word integer array to which the subroutine status is returned.

lun is the logical unit number (ignored if present).

The isb array has the standard meaning defined in section 11.5.2.

11.5.21 SCTI: Initializing a Timer Module

The SCTI FORTRAN subroutine sets a timer module to an initial value.

It is issued as follows:

```
CALL SCTI (imod,ival,[isb],[lun])
```

where: imod is the module number of the timer to be set.

ival is the initial timer value.

isb is a 2-word integer array to which the subroutine status is returned.

lun is the logical unit number.

The isb array has the standard meaning defined in section 11.5.2.

Calls to initialize a counter are valid only if the issuing task has connected a buffer for receiving counter interrupts via a call to CTTI.

11.6 STATUS RETURNS

Table 11-6 lists the error and status conditions that are returned by the UDC11 driver described in this chapter:

Table 11-6
UDC11 Status Returns

Code	Reason
IS.SUC	Sucessful completion The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of samples completed or converted.
IS.PND	I/O request pending The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with zeros.
IE.ABO	Operation aborted The specified I/O operation was cancelled via IO.KIL while still in the I/O queue.
IE.BAD	Bad parameter An illegal specification was supplied for one or more of the device-dependent QIO parameters (words 6-11). For the UDC11, this code indicates an illegal channel number or gain code for the ADU01.

Table 11-6 (Cont.)
UDC11 Status Returns

Code	Reason
IE.BYT	Byte-aligned buffer specified Byte alignment was specified for a buffer but only word alignment is legal for the UDC11. Alternately, the length of a buffer was not an even number of bytes.
IE.CON	Connect error The task attempted to connect to contact or timer interrupts, but the interrupt was already connected to another task. Only one task can be connected to a timer or contact interrupt. Alternately a task which was not connected attempted to disconnect from contact or timer interrupts.
IE.DNR	Device not ready The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation. For the ADU01, this code is returned if an interrupt timeout occurred or the power failed.
IE.IEF	Invalid event flag number The trigger event flag number specified in a connect function was not in the range 1 to 64.
IE.IFC	Illegal function A function code was included in an I/O request that is illegal for the UDC11, or a request to initialize a counter (IO.ITI) was issued by a task that was not connected to receive counter interrupts. The function may also refer to a UDC11 feature which was not specified at system generation.
IE.MOD	Invalid UDC11 module On latching output, the user specified a starting point number which was not legal (defined at system generation) or was not evenly divisible by 16.
IE.OFL	Device off-line The physical device unit associated with the LUN specified in the QIO directive was not on-line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.PRI	Privilege violation The task which issued the request was not privileged to execute that request. For the UDC11, this code indicates that a checkpointable task attempted to connect to timer or contact interrupts.

Table 11-6 (Cont.)
UDC11 Status Returns

Code	Reason
IE.SPC	Illegal address space The specified control, data, or interrupt buffer was partially or totally outside the address space of the issuing task. Alternately, the interrupt buffer was too small for a single data entry (6 words for timer interrupts and 7 words for contact interrupts) or a byte count of zero was specified.

FORTRAN interface values for these status returns are presented in section 11.6.1.

11.6.1 FORTRAN Interface Values

The values listed in Table 11-7 are returned in FORTRAN subroutine calls.

Table 11-7
FORTRAN Interface Values

Status Return	FORTRAN Value
IS.SUC	+01
IS.PND	+00
IE.ABO	+315
IE.ADP	+101
IE.BAD	+301
IE.BYT	+319
IE.DAO	+313
IE.DNR	+303
IE.IEF	+100
IE.IFC	+302
IE.ILU	+99
IE.MOD	+321
IE.ONP	+305
IE.PRI	+316
IE.RSU	+317
IE.SDP	+102
IE.SPC	+306
IE.ULN	+08
IE.UPN	+04

11.7 PROGRAMMING HINTS

This section contains information on important programming considerations relevant to users of the UDC11 driver described in this chapter.

11.7.1 Checkpointable Tasks

Since checkpointable tasks are not allowed to have more than one outstanding I/O request, a task that issues a request to connect to timer or contact interrupts must not be checkpointable.

11.7.2 Numbering Conventions

Numbering is relative. Module numbers start at 0, beginning with the first module of a given type.

Channel numbers also start at 0, with channel 0 as the first channel on the first module of a given type. For the ADU01, channel 8 is the first channel on the second analog output module.

Each IAD-IA module installed in an ICS11 subsystem occupies 120 channels (regardless of the number of multiplexers installed). In this case, channel 120 is the first channel on the second IAD-IA A/D converter.

Point numbers start at 0, with point 0 as the first point on the first module of a given type. For instance, point 20 (octal) is the first point of the second contact sense module (i.e., relative module number 1).

11.7.3 Processing Circular Buffer Entries

Circular buffer entries should be processed in the following sequence.

1. Execute a WAITFOR system directive using the trigger event flag specified in the subroutine called to connect the circular buffer (CTTI or CTDI).

2. Repeatedly call the appropriate subroutine to read the circular buffer until all entries have been obtained, and ipt indicates that the buffer is empty (-1).

3. Perform any other processing and return to step 1.

CHAPTER 12

LABORATORY PERIPHERAL SYSTEMS DRIVERS

12.1 INTRODUCTION

The LPS11 and AR11 Laboratory Peripheral Systems are modular, real-time subsystems used for the acquisition and/or output of laboratory analog data. Table 12-1 compares the LPS11 with the AR11.

Table 12-1
Laboratory Peripheral Systems

	LPS11	AR11
Analog-to-Digital Conversion (with sample and hold circuitry)	12 bits of precision 16-channel multiplexer with gain ranging Maximum of 64 channels without gain ranging	10 bits of precision 16-channel multiplexer without gain ranging
Programmable Real-Time Clock	Yes	Yes
Digital-to-Analog Output	12 bits of precision 10 channels (including display)	10 bits of precision 2 channels (including display)
Display Control	4096 by 4096 dot matrix	1024 by 1024 dot matrix
Digital I/O Option	16 digital points and programmable relays	16 digital points (available with DR11-K option)

At system generation, the user can specify the following:

- Number of A/D channels
- Presence or absence of the gain ranging option (LPSAM-SG) (LPS11 only) and the polarity of each channel (uni- or bi-polar).
- Presence or absence of the external D/A option (LPSVC and LPSDA), and if present, the number of D/A channels.
- Clock preset value

12.1.1 AR11 Laboratory Peripheral System

The AR11 is a one-module, real-time analog subsystem that interfaces to the PDP-11 family of computers via a "hex" small peripheral controller slot. The system is a subset of the LPS11, and as such, enjoys the same degree of flexibility. The AR11 includes a 16-channel, 10-bit A/D converter with sample-and-hold, a programmable real-time clock with one external input, and a display control with two 10-bit D/A converters.

12.1.2 LPS11 Laboratory Peripheral System

The LPS11 is a high-performance, modular, real-time subsystem with the flexibility of serving a variety of applications, including biomedical research, analytical instrumentation, data collection and reduction, monitoring, data logging, industrial testing, engineering, and technical education. The basic subsystem, built in a compact size and designed for easy interface with external instrumentation, includes a 12-bit A/D converter, a programmable real-time clock, with two Schmitt triggers, a display controller with two 12-bit D/A converters, and a 16-bit digital I/O option. Up to nine different option types may be added to the basic package.

12.2 GET LUN INFORMATION MACRO

If a GET LUN INFORMATION system directive is issued for a LUN associated with a Laboratory Peripheral System, word 2 (the first characteristics word) contains all zeros, words 3 and 4 are undefined, and word 5 contains a 16-bit buffer preset value that controls the rate of the real-time clock interrupts, as explained in section 12.6.1.

12.3 QIO MACRO

This section summarizes standard and device-specific QIO functions for the Laboratory Peripheral System drivers.

12.3.1 Standard QIO Function

Table 12-2 lists the standard function of the QIO macro that is valid for the Laboratory Peripheral Systems.

Table 12-2
Standard QIO Function for
Laboratory Peripheral Systems

Format	Function
QIO$C IO.KIL,...	Cancel I/O requests

IO.KIL cancels all queued and in-progress I/O requests.

12.3.2 Device-Specific QIO Functions (Immediate)

Except for IO.STP (see section 12.3.4), all device-specific functions of the QIO macro that are valid for the Laboratory Peripheral Systems are either immediate or synchronous. Each immediate function performs a complete operation, whereas each synchronous function simply initiates an operation synchronized to the real-time clock. Table 12-3 lists the immediate functions.

Table 12-3
Device-Specific Functions for the
Laboratory Peripheral Systems (Immediate)

Format	Function
QIO$C IO.LED,...,<int,num>	Display number in LED lights (LPS11 only)
QIO$C IO.REL,...,<rel,pol>	Latch output relay (LPS11 only)
QIO$C IO.SDI,...,<mask>	Read digital input register
QIO$C IO.SDO,...,<mask,data>	Write digital output register

where:

int	is the 16-bit signed binary integer to display.
num	is the LED digit number where the decimal point is to be placed.
rel	is the relay number (zero or one).
pol	is the polarity (zero for open, nonzero for closed).
mask	is the mask word.
data	is the data word.

The following subsections describe the functions listed above.

12.3.2.1 IO.LED - This LPS11-only function displays a 16-bit signed binary integer in the light-emitting diode (LED) lights. The number is displayed with a leading blank (positive number) or minus sign (negative number) followed by five nonzero-suppressed decimal digits that represent the magnitude of the number. LED digits are numbered from right to left, starting at 1.

The number may be displayed with or without a decimal point. If the parameter num is a number from 1 to 5, then the corresponding LED digit is displayed with a decimal point to the right of the digit. If the LED digit number is not a number from 1 to 5, then no decimal point is displayed.

12.3.2.2 IO.REL - This LPS11-only function opens or closes the programmable relays in the digital I/O status register. Approximately 300 milliseconds are required to open or close a relay. The driver imposes no delays when executing this function. Thus it is the responsibility of the user to insure that adequate time has elapsed between the opening and closing of a relay.

12.3.2.3 IO.SDI - This function reads data qualified by a mask word from the digital input register. The mask word contains a 1 in each bit position from which data is to be read. All other bits are zero-filled and the resulting value is returned in the second I/O status word.

The operation performed is:

RETURN VALUE=MASK.AND.INPUT REGISTER

12.3.2.4 IO.SDO - This function writes data qualified by a mask word into the digital output register. The mask word contains a 1 in each bit position that is to be written. The data word specifies the data to be written in corresponding bit positions.

The operation performed is:

NEW REGISTER=<MASK.AND.DATA>.OR.<<.NOT.MASK>.AND.OLD REGISTER>

12.3.3 Device-Specific QIO Functions (Synchronous)

Table 12-4 lists the synchronous, device-specific functions of the QIO macro that are valid for the Laboratory Peripheral Systems.

Table 12-4
Device-Specific QIO Functions for the
Laboratory Peripheral Systems (Synchronous)

Format	Function
QIO$C IO.ADS,...,<stadd,size,pnt, ticks,bufs,chna>	Initiate A/D sampling
QIO$C IO.HIS,...,<stadd,size,pnt, ticks,bufs>	Initiate histogram sampling (LPS11 only)
QIO$C IO.MDA,...,<stadd,size,pnt, ticks,bufs,chnd>	Initiate D/A output
QIO$C IO.MDI,...,<stadd,size,pnt, ticks,bufs,mask>	Initiate digital input sampling
QIO$C IO.MDO,...,<stadd,size,pnt, ticks,bufs,mask>	Initiate digital output

where:

stadd	is the starting address of the data buffer (must be on a word boundary).
size	is the data buffer size in bytes (must be greater than zero and a multiple of four bytes).
pnt	is the digital point numbers (byte 0 - starting input/output point number; byte 1 - input point number to stop the function).
ticks	is the number of real-time clock ticks between samples or data transfers, as appropriate.
bufs	is the number of data buffers to transfer.
chna	is the analog-to-digital conversion specification. Byte 0 contains the starting channel number. For LPS11 this must be in the range of 0-63; for AR11 the range is 0-15. If the LPS11 gain ranging option is present, the channel number must be in the range of 0-15, and bits 4 and 5 specify the gain code. Byte 1 contains the number of consecutive analog-to-digital channels to sample. For LPS11 this must be in the range of 1-64; for AR11 the range is 1-16.
chnd	is the digital-to-analog output channel specification. Byte 0 contains the starting channel number. For LPS11 this must be in the range of 0-9; for the AR11 the range is 0-1. Byte 1 contains the number of consecutive channels to be output. For LPS11 this must be in the range of 1-10; for AR11 the range is 1-2.
mask	is the mask word.

The following subsections describe the functions listed above.

12.3.3.1 IO.ADS - This function reads one or more A/D channels at precisely timed intervals, with or without auto gain-ranging. If two or more channels are specified, all are sampled at approximately the same time, once per interval.

Sampling may be started when the request is dequeued or when a specified digital input point is set. A digital output point may optionally be set when sampling is started. Sampling may be terminated by a program request (IO.STP or IO.KIL), by the clearing of a digital input point, or by the collection of a specified number of buffers of data.

All input is double-buffered with respect to the user task. Each time a half buffer of data has been collected, the user task is notified (via the setting of an event flag) that data is available to be processed while the driver fills the other half of the buffer. If the user task does not respond quickly enough, a data overrun may result. This occurs if the driver attempts to put another data item in the user buffer when no space is available.

The subfunction modifier bits are identical to those described in section 12.3.3.2. In addition, setting bit 3 to a 1 means LPS11 auto gain-ranging is requested. Bit 3 is ignored for the AR11. If bits 7 and 6 are both set to 1, the digital input point and digital output point number are assumed to be the same.

If LPS11 auto gain-ranging is used, the LPSAM-SG hardware option must be present and specified at system generation. The auto gain-ranging algorithm causes a channel to be sampled at the highest gain at which saturation does not occur. If the gain-ranging option is present and auto gain-ranging is not specified in bit 3 of the subfunction code, then bits 4 and 5 of the starting channel number specify the gain at which samples are to be converted. Gain codes are as follows:

Code	Gain
00	1
01	4
10	16
11	64

Data words written into the user buffer contain the converted value in bits 0-11 and the gain code, as shown below, in bits 12-15:

Code	Gain
0000	1
0001	4
0010	16
0011	64

If the LPSAM-SG option is present, then the band pass filter jumpers must not be clipped. Also, each channel must have been defined as uni- or bi-polar at system generation.

The AR11 always returns data which is equivalent to an LPS11 gain of 1. Channel polarity must always be specified for the AR11 at system generation since this operation is software selectable.

12.3.3.2 IO.HIS - This LPS11-only function measures the elapsed time between a series of events by means of Schmitt trigger one. Each time a sample is to be taken, a counter is incremented and Schmitt trigger one is tested. If it has fired, then the counter is written into the user buffer and reset to zero. Thus the data item returned to the user is the number of sample intervals between Schmitt trigger firings.

If the counter overflows before Schmitt trigger one fires, then a zero value is written into the user buffer. Sampling may be started and stopped as described in section 12.3.3.1. All input is double-buffered with respect to the user task.

The subfunction modifier bits appear below. A setting of 1 indicates the action listed in the right-hand column.

Bit	Meaning
0-3	Unused
4	Stop on number of buffers
5	Stop on digital input point clear
6	Set digital output point at start of operation
7	Start on digital input point set (a zero specification means start immediately)

12.3.3.3 IO.MDA - This function writes data into one or more external D/A converters at precisely timed intervals. If two or more channels are specified, all are written at approximately the same time, once per interval. Output may be started or stopped as described in section 12.3.3.1. All output is double-buffered with respect to the user task.

D/A converters 0 and 1 correspond to the X and Y registers of the display control. Note that there are no specific driver functions to set the display status register. This is reserved for the user. D/A converters 2 through 9 correspond to the LPS11, LPSDA external D/A option.

The subfunction modifier bits are identical to those described in section 12.3.3.2.

12.3.3.4 IO.MDI - This function provides the capability to read data qualified by a mask word from the digital input register at precisely timed intervals. Sampling may be started and stopped as described in section 12.3.3.1. All input is double-buffered with respect to the user task.

The mask word contains a 1 in each bit position from which data is to be read. All other bits are zero.

The subfunction modifier bits are identical to those described in section 12.3.3.2.

12.3.3.5 IO.MDO - This function writes data qualified by a mask word into the digital output register at precisely timed intervals. Output may be started and stopped as described in section 12.3.3.1. All output is double-buffered with respect to the user task.

The subfunction modifier bits are identical to those described in section 12.3.3.2.

12.3.4 Device-Specific QIO Function (IO.STP)

Table 12-5 lists the device-specific function of the QIO macro, which is valid for the Laboratory Peripheral Systems.

Table 12-5
Device-Specific QIO Function for the
Laboratory Peripheral Systems (IO.STP)

Format	Function
QIO$C IO.STP,...,<stadd>	Stop in-progress request

where: stadd is the buffer address of the function to stop (must be the same as the address specified in the initiating request).

12.3.4.1 IO.STP - IO.STP stops a single in-progress synchronous request. It is unlike IO.KIL in that it only cancels the specified request, whereas IO.KIL cancels all requests.

12.4 FORTRAN INTERFACE

A collection of FORTRAN-callable subroutines provide FORTRAN programs access to the Laboratory Peripheral Systems. These routines are described in this section.

Some of these routines may be called from FORTRAN as either subroutines or functions. All are reentrant and may be placed in a resident library.

12.4.1 The isb Status Array

The isb (I/O status block) parameter is a two-word integer array that contains the status of the FORTRAN call, in accordance with ISA convention. This array serves two purposes:

1. It is the 2-word I/O status block to which the driver returns an I/O status code on completion of an I/O operation.

2. The first word of the isb receives a status code from the FORTRAN interface in ISA-compatible format, with the exception of the I/O pending condition, which is indicated by a status of zero. The ISA standard code for this condition is +2.

The meaning of its contents varies, depending on the FORTRAN call that has been executed, but Table 12-6 lists certain general principles that apply. The sections describing individual subroutines provide more details.

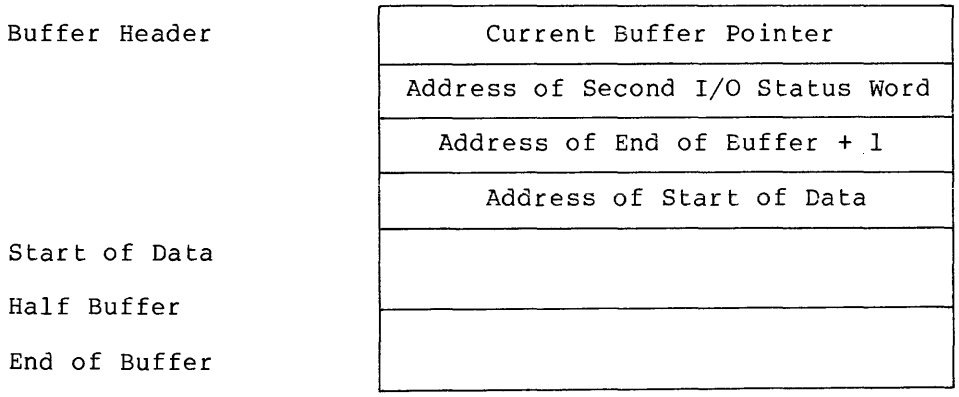
Table 12-6
Contents of First Word of isb

Contents	Meaning
isb(1) = 0	Operation pending; I/O in progress
isb(1) = 1	Successful completion
isb(1) = 3	Interface subroutine unable to generate QIO directive, or illegal time or buffer value
3 <= isb(1) $\leq$ 300	QIO directive rejected and actual error code = -(isb(1) - 3)
isb(1) > 300	Driver rejected request and actual error code = -(isb(1) - 300)

FORTRAN interface routines depend on asynchronous system traps to set their status. Thus, if the trap mechanism is disabled, proper status cannot be set.

12.4.2 Synchronous Subroutines

RTS, DRS, HIST (LPS11 only), SDO, and SDAC are FORTRAN subroutines that initiate synchronous functions. When they are used, the appropriate laboratory peripheral system driver and the FORTRAN program communicate by means of a caller-specified data buffer of the following format:



The buffer header is initialized when the synchronous function initiation routine is called. The length of the buffer must be even and greater than or equal to six. An even length is required so that the buffer is exactly divisible into half buffers.

The drivers perform double buffering within the half buffers. Each time a driver fills or empties a half buffer, it sets a user-specified event flag to notify the user task that more data is available or needed. The user task responds by putting more data into the buffer or by removing the data now available.

If the user task does not respond quickly enough, a data overrun may result. This occurs if the driver attempts to put another data item

in the user buffer when no space is available (i.e., the buffer is full of data) or if the driver attempts to obtain the next data item from the user buffer when none is available (i.e., the buffer is empty).

All synchronous functions may be initiated immediately or when a specified digital input point is set (i.e., a start button is pushed).

They may be terminated by any combination of a program request, the processing of the required number of full buffers of data, or the clearing of a specified digital input point (i.e., a stop button is pushed). A digital output point may also optionally be set at the start of a synchronous function. This could be used, for example, as a signal to start a test instrument.

12.4.3 FORTRAN Subroutine Summary

Table 12-7 lists the FORTRAN interface subroutines supported for the Laboratory Peripheral Systems under RSX-11M. S and F indicate whether they can be called as subroutines or functions.

Table 12-7
FORTRAN Interface Subroutines for Laboratory Peripheral Systems

Subroutine	Function
ADC	Read a single A/D channel (F,S)
ADJLPS	Adjust buffer pointers (S)
ASARLN	Assign a LUN to AR0: (S)
ASLSLN	Assign a LUN to LS0: (S)
CVSWG	Convert a switch gain A/D value to floating-point (F)
DRS	Initiate synchronous digital input sampling (S)
HIST	Initiate histogram sampling (S) (LPS11 only)
IDIR	Read digital input (F,S)
IDOR	Write digital output (F,S)
IRDB	Read data from a synchronous function input buffer (F,S)
LED	Display number in LED lights (S) (LPS11 only)
LPSTP	Stop an in-progress synchronous function (S)
PUTD	Put data into a synchronous function output buffer (S)
RELAY	Latch an output relay (S) (LPS11 only)
RTS	Initiate synchronous A/D sampling (S)

Table 12-7 (Cont.)
FORTRAN Interface Subroutines for Laboratory Peripheral Systems

Subroutine	Function
SDAC	Initiate synchronous D/A output (S)
SDO	Initiate synchronous digital output (S)

The following subsections briefly describe the function and format of each FORTRAN subroutine call.

12.4.4 ADC: Reading a Single A/D Channel

The ADC FORTRAN subroutine or function reads a single converted value from an A/D channel. If the gain-ranging option is present in the LPS11 hardware, the channel may be converted at a specific gain or the driver can select the best gain (the gain providing the most significance). The converted value is returned as a normalized floating-point number. The call is issued as follows:

```
CALL ADC (ichan,[var],[igain],[isb])
```

where:

ichan — specifies the A/D channel to be converted.

var — is a floating-point variable that receives the converted value in floating-point format.

igain — specifies the gain at which the specified A/D channel is to be converted. The default is 1. If specified, igain may have the following values:

igain	Gain
0	Auto gain-ranging (driver selects gain that provides most significance)
1	1
2	4
3	16
4	64

A gain of 1 is always used by the AR11 driver.

isb — is a 2-word integer array to which the subroutine status is returned.

The isb array has the standard meaning described in section 12.4.1.

When the function form of the call is used, the value of the function is the same as that returned in var. If this value is negative, an error has occurred during the A/D conversion (see section 12.5.3). Otherwise, this value is a floating-point number calculated from the following formula:

```
var = (64 * converted value) / conversion gain
```

12.4.5 ADJLPS: Adjusting Buffer Pointers

The ADJLPS FORTRAN subroutine adjusts buffer pointers for a buffer that a laboratory peripheral system driver is either synchronously filling or emptying. It is usually called when indexing is being used for direct access to the data in a buffer.

When data in a buffer is to be processed only once, the IRDB and PUTD routines may be used. In some cases, however, it is useful to leave data in the buffer until processing is complete. The user program may process the data directly, then call ADJLPS to free half the buffer. Use of the routine for synchronous output functions is quite similar. When a half buffer of data is ready for output, ADJLPS is called to make the half buffer available.

When ADJLPS is used for either input or output, care must be taken to insure that the program stays in sync with the driver. If the program loses its position with respect to the driver, the function must be stopped and restarted. An attempt to over-adjust will cause a 3 to be returned in isb(1) and no adjustment to take place.

The call is issued as follows:

```
CALL ADJLPS (ibuf,iadj,[isb])
```

where:

ibuf	is an integer array which was previously specified in a synchronous input or output function.
iadj	specifies the adjustment to be applied to the buffer pointers. For an input function this specifies the number of data values that have been removed from the data buffer. For an output function this specifies the number of data values that have been put into the data buffer.
isb	is a 2-word integer array to which the subroutine status is returned.

The isb array has the standard meaning described in section 12.4.1.

12.4.6 ASLSLN: Assigning a LUN to LS0:

The ASLSLN FORTRAN subroutine assigns a logical unit number (LUN) to the LPS11. It must be called prior to executing any other Laboratory Peripheral Systems FORTRAN function or subroutine. Subsequent calls to other interface routines then implicitly reference the LPS11 via the LUN assigned.

The call is issued as follows:

```
CALL ASLSLN (lun,[isb])
```

where:

lun	is the number of the LUN to be assigned to LS0:
isb	is a 2-word integer array to which the subroutine status is returned.

The isb array has the standard meaning described in section 12.4.1.

12.4.7 ASARLN: Assigning a LUN to AR0:

The ASARLN FORTRAN subroutine assigns a logical unit number (LUN) to the AR11. It must be called prior to executing any other laboratory peripheral system FORTRAN function or subroutine. Subsequent calls to other interface routines then implicitly reference the AR11 via the LUN assigned.

The call is issued as follows:

```
CALL ASARLN (lun,[isb])
```

where:

lun	is the number of the LUN to be assigned to AR0:
isb	is a 2-word integer array to which the subroutine status is returned.

The isb array has the standard meaning described in section 12.4.1.

12.4.8 CVSWG: Converting a Switch Gain A/D Value to Floating-Point

The CVSWG FORTRAN subroutine converts an A/D value from a synchronous A/D sampling function to a floating-point number. Each data item returned by a laboratory peripheral system driver consists of a gain code and converted value packed in a single word (see section 12.3.3.1). This form is not readily usable by FORTRAN, but is much more efficient than converting each value to floating-point in the driver. This routine unpacks the gain code and value, then converts the result to a floating-point number. It may be conveniently used in conjunction with the IRDB routine (see section 12.4.13).

The call is issued as follows:

```
CVSWG (ival)
```

where: ival is the value to be converted to floating point. Its format must be that returned by a synchronous A/D sampling function. The conversion is performed according to the following formula:

var = (64 * converted value)/conversion gain

For the various gain codes,

var = x * converted value

as shown below:

Gain	x
1	64
4	16
16	4
64	1

12.4.9 DRS: Initiating Synchronous Digital Input Sampling

The DRS FORTRAN subroutine reads data qualified by a mask word from the digital input register at precisely timed intervals. Sampling may be started or stopped as for RTS (see section 12.4.18) and all input is double-buffered with respect to the user task. Data may be sequentially retrieved from the data buffer via the IRDB routine (see section 12.4.12), or the ADJLPS routine (see section 12.4.5) may be used in conjunction with direct access to the input data. The call is issued as follows:

```
CALL DRS (ibuf,ilen,imode,irate,iefn,imask,isb,[nbuf],
          [istart],[istop])
```

where: ibuf is an integer array that is to receive the input data values.

ilen specifies the length of ibuf (must be even and greater than or equal to six).

imode specifies the start, stop, and sampling mode. Its value is encoded by adding together the appropriate function selection values shown below.

Function Selection Value	Meaning
128	Start on digital input point set
64	Set digital output point at start
32	Stop on digital input point clear
16	Stop on number of buffers

Thus a value of 192 for imode specifies:

- The sampling is to be started when a specified digital input point is set.
- A digital output point is to be set when sampling is started.
- Sampling will be stopped via a program request.

irate is a 2-word integer array that specifies the time interval between digital input samples. The first word specifies the interval units as follows:

irate(1)	Unit
1	Real-time clock ticks
2	Milliseconds
3	Seconds
4	Minutes

The second word specifies the interval magnitude as a 16-bit unsigned integer.

iefn specifies the number of the event flag that is to be set each time a half buffer of data has been collected.

imask specifies the digital input points to be read.

isb is a 2-word integer array to which the subroutine status is returned.

nbuf specifies the number of buffers of data to be collected. It is needed only if a function selection value of 16 has been added into imode.

istart — specifies the digital input pointer number to be used to trigger sampling and/or the digital output point number to be set when sampling is started. It is needed only if a function selection value of 128 or 64 has been added into imode.

istop — specifies the digital input point number to be used to stop sampling. It is needed only if a function selection value of 32 has been added into imode.

When sampling is in progress, the first word of the isb array is zero and the second word contains the number of data values currently in the buffer.

12.4.10 HIST: Initiating Histogram Sampling (LPS11 only)

The HIST FORTRAN subroutine measures the elapsed time between a series of events via Schmitt trigger one.

Each time a sample is to be taken, a counter is incremented and Schmitt trigger one is tested. If it has fired, then the counter is written into the user buffer and the counter is reset to zero. Thus the data returned to the user is the number of sample intervals between Schmitt trigger firings. If the counter overflows before Schmitt trigger one fires, a zero value is written into the user buffer. Sampling may be started and stopped as for RTS (see section 12.4.18) and all input is double-buffered with respect to the user task. The call is issued as follows:

```
CALL HIST (ibuf,ilen,imode,irate,iefn,isb,[nbuf],
           [istart],[istop])
```

where:

ibuf — is an integer array that is to receive the input data values.

ilen — specifies the length of ibuf (must be even and greater than or equal to six).

imode — specifies the start, stop and sampling mode. Its value is encoded by adding the appropriate function selection values shown below:

Function Selection Value	Meaning
128	Start of digital input point set
64	Set digital output point at start

32	Stop on digital input point clear
16	Stop on number of buffers

irate is a 2-word integer array that specifies the time interval between samples. The first word specifies the interval units as follows:

irate(1)	Unit
1	Real-time clock ticks
2	Milliseconds
3	Seconds
4	Minutes

The second word specifies the interval magnitude as a 16-bit signed integer.

iefn specifies the number of the event flag that is to be set each time a half buffer of data has been collected.

isb is a 2-word integer array to which the subroutine status is returned.

nbuf specifies the number of buffers of data to be collected. It is needed only if a function selection value of 16 has been added into imode.

istart specifies the digital input point number to be used to trigger sampling and/or the digital output point number to be set when sampling is started. It is needed only if a function selection value of 128 or 64 has been added into imode.

istop specifies the digital input point number to be used to stop sampling. It is needed only if a function selection value of 32 has been added into imode.

The isb array has the standard meaning described in section 12.4.1.

When sampling is in progress, the first word of the isb array is zero and the second word contains the number of data values currently in the buffer.

12.4.11 IDIR: Reading Digital Input

The IDIR FORTRAN subroutine or function reads the digital input register as an unsigned binary integer or as four binary-coded decimal (BCD) digits. In the latter case, the BCD digits are converted to a binary integer before the value is returned to the caller. The call is issued as follows:

```
CALL IDIR (imode,[ival],[isb])
```

where:

imode	specifies the mode in which the digital input register is to be read. If imode equals zero, then the digital input register is read as four BCD digits and converted to a binary integer. Otherwise it is read as a 16-bit unsigned binary integer.
ival	is a variable that receives the value read.
isb	is a 2-word integer array to which the subroutine status is returned.

The isb array has the standard meaning described in section 12.4.1.

When the function form of the call is used, the value of the function is the same as that returned in ival.

12.4.12 IDOR: Writing Digital Output

The IDOR FORTRAN subroutine or function clears or sets bits in the digital output register. The caller provides a mask word and output mode. Bits in the digital output registers corresponding to the bits specified in the mask word are either set or cleared according to the specified mode. The call is issued as follows:

```
CALL IDOR (imode,imask,[newval],[isb])
```

where:

imode	specifies whether the bits specified by imask are to be cleared or set in the digital output register. If imode equals zero, then the bits are to be cleared. Otherwise they are to be set.
imask	specifies the bits to be cleared or set in the digital output register. It may be conveniently specified as an octal constant.
newval	is a variable that receives the updated (actual) value written into the digital output register.
isb	is a 2-word integer array to which the subroutine status is returned.

The isb array has the standard meaning described in section 12.4.1.

When the function form of the call is used, the value of the function is the same as that returned in newval.

12.4.13 IRDB: Reading Data from an Input Buffer

The IRDB FORTRAN subroutine or function retrieves data sequentially from a buffer that a laboratory peripheral system driver is synchronously filling. If no data is available when the call is executed, the contents of the next location in the data buffer are returned without updating the buffer pointers. The call is issued as follows:

```
CALL IRDB (ibuf,[ival])
```

where:

ibuf	is an integer array which was previously specified in a synchronous input sampling request (i.e., DRS, HIST, or RTS).
ival	is a variable that receives the next value in the data buffer.

When the function form of the call is used, the value of the function is the same as that returned in ival.

12.4.14 LED: Displaying in LED Lights (LPS11 only)

The LED FORTRAN subroutine displays a 16-bit signed binary integer in the LED lights. The number is displayed with a leading blank (positive number) or minus (negative number), followed by five non-zero suppressed decimal digits that represent the magnitude of the number. LED digits are numbered right to left starting at 1 and continuing to 5. The number may be displayed with or without a decimal point. The call is issued as follows:

```
CALL LED (ival,[idec],[isb])
```

where:

ival	is the variable whose value is to be displayed.
idec	specifies the position of the decimal point. A value of 1 to 5 specifies that a decimal point is to be displayed. All other values specify that no decimal point is to be displayed.
isb	is a 2-word integer array to which the subroutine status is returned.

The isb array has the standard meaning described in section 12.4.1.

For example, the following call

```
CALL LED (-55,2)
```

would cause -0005.5 to be displayed in the LED lights.

12.4.15 LPSTP: Stopping an In-Progress Synchronous Function

The LPSTP FORTRAN subroutine selectively stops a single synchronous request. The call is issued as follows:

```
CALL LPSTP (ibuf)
```

where: ibuf is an integer array that specifies a buffer that was previously specified in a synchronous initiation request.

12.4.16 PUTD: Putting a Data Item into an Output Buffer

The PUTD FORTRAN subroutine puts data sequentially into a buffer that a laboratory peripheral system driver is synchronously emptying. If no free space is available, no operation is performed. The call is issued as follows:

```
CALL PUTD (ibuf,ival)
```

where:

ibuf is an integer array which was previously specified in a synchronous output request (SDO or SDAC).

ival is a variable whose value is to be placed in the next free location in the data buffer.

12.4.17 RELAY: Latching an Output Relay (LPS11 only)

The RELAY FORTRAN subroutine opens or closes the LPS11 relays. The call is issued as follows:

```
CALL RELAY (irel,istate,[isb])
```

where:

irel specifies which relay is to be opened or closed (one for relay one, two for relay two).

istate specifies whether the relay is to be opened or closed. If istate equals zero, the relay is to be opened. Otherwise it is to be closed.

isb is a 2-word integer array to which the subroutine status is returned.

The isb array has the standard meaning described in section 12.4.1.

12.4.18 RTS: Initiating Synchronous A/D Sampling

The RTS FORTRAN subroutine reads one or more A/D channels at precisely timed intervals, with or without auto gain-ranging. The auto gain-ranging algorithm (LPS11 only) causes the channels to be sampled at the highest gain at which saturation does not occur.

Sampling may be started when the interface subroutine is called or when a specified digital input point is set. A digital output point may optionally be set when sampling is started. Sampling may be terminated by a program request (stop-in-progress request or kill I/O), the clearing of a digital input point, or the collection of a specified number of buffers of data.

All input is double-buffered with respect to the user task. Each time a half buffer of data has been collected, the user task is notified (via the setting of an event flag) that data is available to be processed while the driver fills the other half of the buffer. Data may be sequentially retrieved from the data buffer via the IRDB routine (see section 12.4.12, or the ADJLPS routine (see section 12.4.5) may be used in conjunction with direct access to the input data. The call is issued as follows:

```
CALL RTS (ibuf,ilen,imode,irate,iefn,ichan,nchan,isb,
          [nbuf],[istart],[istop])
```

where:

ibuf — is an integer array that is to receive the converted data values.

ilen — specifies the length of ibuf (must be even and greater than or equal to six).

imode — specifies the start, stop, and sampling mode. Its value is encoded by adding together the appropriate function selection values as shown below:

Function Selection Value	Meaning
128	Start on digital input point set
64	Set digital output point at start
32	Stop on digital input point clear
16	Stop on number of buffers
8	Auto gain-ranging (LPS11 only)

irate — is a 2-word integer array that specifies the time interval between A/D samples. The first word specifies the interval unit as follows:

irate(1)	Unit
1	Real-time clock ticks
2	Milliseconds
3	Seconds
4	Minutes

The second word specifies the interval magnitude as a 16-bit unsigned integer.

iefn — specifies the number of the event flag that is to be set each time a half buffer of data has been collected.

ichan — specifies the starting A/D channel of the block of channels to be sampled synchronously (must be between 0 and 63 for LPS11 and between 0 and 15 for AR11).

nchan — specifies the number of A/D channels to be sampled (must be between 1 and 64 for LPS11 and between 1 and 16 for AR11).

isb — is a 2-word integer array to which the subroutine status is returned.

nbuf — specifies the number of buffers of data that are to be collected. It is needed only if a function selection value of 16 has been added into imode.

istart — specifies the digital input point number to be used to trigger sampling and/or the digital output point number to be set when sampling is started. It is needed only if a function selection value of 128 or 64 has been added into imode.

istop — specifies the digital input point number to be used to stop sampling. It is needed only if a function selection value of 32 has been added into imode.

The values listed for ichan and nchan above, are the maximum allowable for each of the devices. In practice, they are constrained by the number of channels available as specified during SYSGEN.

The isb parameter has the standard meaning described in section 12.4.1.

When sampling is in progress, the first word of the isb array is zero and the second word contains the number of data values currently in the buffer.

12.4.19 SDAC: Initiating Synchronous D/A Output

The SDAC FORTRAN subroutine writes data into one or more external D/A converters at precisely timed intervals. Output may be started and stopped as for RTS (see section 12.4.18) and all input is double-buffered with respect to the user task. One full buffer of data must be available when synchronous output is initiated.

After SDAC has initiated output and the specified event flag has been set to notify the task that free buffer space is available, the PUTD routine (see section 12.4.16) may be used to put data values sequentially into the output data buffer. The ADJLPS routine (see section 12.4.5) may be used in conjunction with direct access to the output data buffer. The SDAC call is issued as follows:

```
CALL SDAC (ibuf,ilen,imode,irate,iefn,ichan,nchan,isb,
           [nbuf],[istart],[istop])
```

where:

ibuf — is an integer array that contains the output data values.

ilen — specifies the length of ibuf (must be even and greater than or equal to six).

imode — specifies the start, stop and sampling mode. Its value is encoded by adding together the appropriate function selection values as shown below:

Function Selection Values	Meaning
128	Start on digital input point set
64	Set digital output point at start
32	Stop on digital input point clear
16	Stop on number of buffers

irate — is a 2-word integer array that specifies the time interval between D/A outputs. The first word specifies the interval units as follows:

irate(1)	Unit
1	Real-time clock ticks
2	Milliseconds
3	Seconds
4	Minutes

The second word specifies the interval magnitude as a 16-bit unsigned integer.

iefn	specifies the number of the event flag that is to be set each time a half buffer of data has been output.
ichan	specifies the starting D/A channel of the block of channels to be written into synchronously (must be between 0 and 9 for LPS11 and be 0 or 1 for AR11).
nchan	specifies the number of D/A channels to be written into (must be between 1 and 10 for LPS11 and be 1 or 2 for AR11).
isb	is a 2-word integer array to which the subroutine status is returned.
nbuf	specifies the number of buffers of data to be output. It is needed only if a function selection value of 16 has been added into imode.
istart	specifies the digital input point number to be used to trigger sampling and/or the digital output point number to be set when sampling is started. It is needed only if a function selection value of 128 or 64 has been added into imode.
istop	specifies the digital input point number to be used to stop sampling. It is needed only if a function selection value of 32 has been added into imode.

The isb array has the standard meaning described in section 12.4.1.

When sampling is in progress, the first word of the isb array is zero and the second word contains the number of free positions in the buffer.

12.4.20 SDO: Initiating Synchronous Digital Output

The SDO FORTRAN subroutine writes data qualified by a mask word into the digital output register at precisely timed intervals. Sampling may be started and stopped as for RTS (see section 12.4.18) and all input is double-buffered with respect to the user task. One full buffer of data must be available when output is initiated.

After SDO has initiated output and the specified event flag has been set to notify the task that free buffer space is available, the PUTD routine (see section 12.4.16) may be used to put data values sequentially into the output data buffer. The ADJLPS routine (see section 12.4.5) may be used in conjunction with direct access to the output data buffer. The SDO call is issued as follows:

```
CALL SDO (ibuf,ilen,imode,irate,iefn,imask,isb,[nbuf],
          [istart],[istop])
```

where:

ibuf — is an integer array that contains the digital output values.

ilen — specifies the length of ibuf (must be even and greater than or equal to six).

imode — specifies the start, stop, and sampling mode. Its value is encoded by adding together the appropriate function selection values as shown below:

Function Selection Value	Meaning
128	Start on digital input point set
64	Set digital output point at start
32	Stop on digital input point clear
16	Stop on number of buffers

irate — is a 2-word integer array that specifies the time interval between digital outputs. The first word specifies the interval units as follows:

irate(1)	Unit
1	Real-time clock ticks
2	Milliseconds
3	Seconds
4	Minutes

The second word specifies the interval magnitude as a 16-bit unsigned integer.

iefn — specifies the number of the event flag that is to be set each time a half buffer of data has been output.

imask — specifies the digital output points that are to be written. It may be conveniently specified as an octal constant.

isb — is a 2-word integer array to which the subroutine status is returned.

nbuf — specifies the number of buffers of data to be output. It is needed only if a function selection value of 16 has been added into imode.

istart — specifies the digital input point number to be used to trigger sampling and/or the digital output point number to be set when sampling is started. It is needed only if a function selection value of 128 or 64 has been added into imode.

istop — specifies the digital input point number to be used to stop sampling. It is needed if a function selection value of 32 has been added into imode.

The isb parameter has the standard meaning described in section 12.4.1.

When sampling is in progress, the first word of the isb array is zero and the second word contains the number of free positions in the buffer.

12.5 STATUS RETURNS

The error and status conditions listed in Table 12-8 are returned by the Laboratory Peripheral System drivers described in this chapter.

Table 12-8
Laboratory Peripheral Systems Status Returns

Code	Reason
IS.SUC	Successful completion The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of data values processed.
IS.PND	I/O request pending The operation specified in the QIO directive has not yet been completed.
IE.ABO	Operation aborted The specified I/O operation was canceled (via IO.KIL or IO.STP) while in progress.
IE.BAD	Bad parameter An illegal specification was supplied for one or more of the device-dependent QIO parameters (words 6-11). The second I/O status word is filled with zeros.

Table 12-8 (Cont.)
Laboratory Peripheral Systems Status Returns

Code	Reason
IE.BYT	Byte-aligned buffer specified Byte alignment was specified for a data buffer but only word alignment is legal for Laboratory Peripheral Systems. Alternately, the length of a buffer is not an even number of bytes.
IE.DAO	Data overrun For Laboratory Peripheral Systems, the driver attempted to get a value from the user buffer when none was available or attempted to put a value in the user buffer when no space was available.
IE.DNR	Device not ready The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation. For Laboratory Peripheral Systems, this code is returned if a device time-out occurs while a function is in progress. The second I/O status word contains the number of free positions in the buffer, as appropriate.
IE.IEF	Invalid event flag number An invalid event flag number was specified in a synchronous function (i.e., an event flag number that was not in the range 1 to 64).
IE.IFC	Illegal function A function code was included in an I/O request that is illegal for the LPS11 or AR11.
IE.NOD	Insufficient buffer space Dynamic storage space has been depleted, and there is insufficient buffer space available to allocate a secondary control block for a synchronous function.
IE.OFL	Device off-line The physical device unit associated with the LUN specified in the QIO directive was not on-line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.

Table 12-8 (Cont.)
Laboratory Peripheral Systems Status Returns

Code	Reason
IE.ONP	Option not present An option dependent function or subfunction was requested, and the required feature was not specified at system generation. For example the gain-ranging option or D/A option is not present. The second I/O status word contains zeros.
IE.PRI	Privilege violation The task which issued the request was not privileged to execute that request. For Laboratory Peripheral Systems, a checkpointable task attempted to execute a synchronous sampling function.
IE.RSU	Resource in use A resource needed by the function requested in the QIO directive was being used (see section 12.5.1).
IE.SPC	Illegal address space The buffer specified for a read or write request was partially or totally outside the address space of the issuing task. Alternately a byte count of zero was specified. The second I/O status word contains zeros.

FORTRAN interface values for these status returns are presented in section 12.5.4.

12.5.1 IE.RSU

IE.RSU is returned if a function requests a resource that is currently being used. The requesting task may repeat the request at a later time or take any alternative action required.

Because certain functions do not need such resources, they never cause this code to be returned. Other functions return this code under the following conditions:

Function	When IE.RSU Is Returned
IO.SDO	One or more specified digital output bits are in use
IO.ADS	Digital output point (if specified) is in use
IO.HIS	Digital output point (if specified) is in use
IO.MDA	Digital output point (if specified) is in use

IO.MDI — Digital output point (if specified) or digital input points to be sampled are in use

IO.MDO — Digital output point (if specified) or output bits to be written are in use

The following components of the Laboratory Peripheral Systems are each considered a single resource:

Resource	When Sharable
The A/D Converter and clock	Always sharable.
Each bit in the digital output register	Never sharable.
Each bit in the digital input register	Always sharable when used by IO.SDI or for start/stop conditions (specified in subfunction modifier bits), even when in use by another function; when specified by a synchronous digital input function, not sharable with another such function.

Each resource is allocated on a first-come-first-served basis (i.e., when a conflict arises, the most recent request is rejected with a status of IE.RSU).

12.5.2 Second I/O Status Word

On successful completion of a function specified in a QIO macro call, the IS.SUC code is returned to the first word of the I/O status block

Table 12-9 lists the contents of the second word of the status block, on successful completion for each function.

Table 12-9
Returns to Second Word of I/O Status Block

Successful Function	Contents of Second Word
IO.KIL	Number of data values before I/O was canceled
IO.LED	Zero
IO.REL	Zero
IO.SDI	Masked value read from digital input register
IO.SDO	Updated value written into digital output register

Table 12-9 (Cont.)
Returns to Second Word of I/O Status Block

Successful Function	Contents of Second Word
IO.ADS	Number of data values remaining in buffer
IO.HIS	Number of data values remaining in buffer
IO.MDA	Number of free positions in buffer
IO.MDI	Number of data values remaining in buffer
IO.MDO	Number of free positions in buffer
IO.STP	Zero

When IE.BAD is returned, the second I/O status word contains zero. Laboratory Peripheral Systems drivers return the IE.BAD code under the following conditions:

Function	When IE.BAD is Returned
IO.REL	Relay number not 0 or 1
IO.ADS IO.MDA	No I/O status block, illegal digital I/O point number, or illegal channel number
IO.HIS IO.MDI IO.MDO	No I/O status block or illegal digital I/O point number

12.5.3 IO.ADS and ADC Errors

While IO.ADS or the ADC FORTRAN subroutine is converting a sample, two error conditions may arise. Both of these conditions are reported to the user by placing illegal values in the data buffer. A -1 (177777 octal) is placed in the buffer if an A/D conversion does not complete within 30 microseconds. A -2 (177776 octal) is placed in the buffer if an error occurs during an A/D conversion (LPS11 only).

12.5.4 FORTRAN Interface Values

The values listed in Table 12-10 are returned in FORTRAN subroutine calls.

Table 12-10
FORTRAN Interface Values

Status Return	FORTRAN Value
IS.SUC	+01
IS.PND	+00
IE.ABO	+315
IE.ADP	+101
IE.ALN	+334
IE.BAD	+301
IE.BYT	+319
IE.DAO	+313
IE.DNR	+303
IE.IEF	+100
IE.IFC	+302
IE.ILU	+99
IE.NOD	+323
IE.OFL	+365
IE.ONP	+305
IE.PRI	+316
IE.RSU	+317
IE.SDP	+102
IE.SPC	+306
IE.ULN	+08
IE.UPN	+04

12.6 PROGRAMMING HINTS

This section contains information on important programming considerations relevant to users of the Laboratory Peripheral Systems drivers described in this chapter.

12.6.1 The LPS11/AR11 Clock and Sampling Rates

The basic real-time clock frequency (count rate) for all synchronous functions is always 10KHz. Device characteristics word 4 contains a 16-bit buffer preset value, set dynamically or at system generation, that controls the rate of "ticks" (i.e., the rate at which the clock interrupts). The quotient that results when this value is divided into 10KHz is the rate of "ticks". For example, if this value is 2, the "tick" rate is 5KHz. The user may use a GET LUN INFORMATION system directive to examine the value and a SET /BUF MCR function to modify it while the system is running.

The ticks parameter in a synchronous function specifies the number of "ticks" between samples or data transfers. The value of ticks is a 16-bit number. Thus 65,536 discrete sampling frequencies are possible for each of 65,536 different "tick" rates. This provides a maximum single-channel sample rate of 1 sample every 100 microseconds (possible in hardware but impractical in software) and a minimum of 1 sample every 429,495 seconds. A single-channel rate greater than 2KHz is possible, but not recommended.

The figures below represent initial timing tests run under RSX-11M. It should be noted that no computation was performed on the data other than continuously removing it from or inserting it into the data buffer.

The following data is for the LPS11 on a PDP-11/40 with memory management and no gain-ranging option.

Analog rates:

1 request for 1 channel at 2.5KHz

1 request for 2 channels at 2.0KHz (aggregate 4KHz)

2 requests for 1 channel at 2.0KHz (aggregate 4KHz)

Digital rates:

1 request for 2 channels at 2.5KHz (aggregate 5KHz)

The following data is for the AR11 on a PDP-11/40 with no memory management, no digital I/O option, and no uni-polar sampling.

Analog rates:

1 request for 1 channel at 3.3KHz

1 request for 2 channels at 2.5KHz (aggregate 5.0KHz)

2 requests for 1 channel at 2.5KHz (aggregate 5.0KHz)

Digital rate:

2 requests for 2 channels at 3.3KHz (aggregate 6.6KHz)

12.6.2 Importance of the I/O Status Block

An I/O status block must be specified with every synchronous function. If the first I/O status word is nonzero, the request has been terminated and the value indicates the reason for termination. Otherwise, the request is in progress, and the second I/O status word contains the number of data values remaining in the buffer (or the number of free positions in the buffer for IO.MDA and IO.MDO).

12.6.3 Buffer Management

The buffer unload protocol for synchronous input functions is described below. The user constructs a five-word block that contains the following:

```
IOSB:     .BLKW    2           ; I/O STATUS DOUBLE-WORD
CURPT:    .WORD    BUFFER      ; ADDRESS OF BUFFER
LSTPT:    .WORD    BUFFER+n    ; ADDRESS OF END OF BUFFER
FSTPT:    .WORD    BUFFER      ; ADDRESS OF BUFFER
```

Two of these words are required by the driver (I/O status block) and the remaining three by the user to unload data values from the buffer.

The user then issues the I/O request with the appropriate parameters and the address of the above block as the I/O status block. The QIO directive zeros both I/O status words to initialize them.

If the driver accepts the request, it sets up a write pointer to the first word in the user buffer. Thus the user has a buffer read pointer and the driver has a buffer write pointer. The user and the driver share the second I/O status word, which is the number of data words in the buffer that contain data.

Each time the driver attempts to put a sample value into the buffer, it increments the second I/O status word and compares the result with the size of the buffer. If the result is greater, buffer overrun has occurred and the request is terminated. Otherwise the value is stored in the buffer at the address specified by the driver's write pointer and the write pointer is updated.

If the value stored in the user buffer fills half of the buffer, the event flag specified in the I/O request is set in order to notify the user that a half buffer of data is available to be processed. Each time the user task is awakened, it should execute the following code:

```
5$:     CLEF$S    #EFN          ;CLEAR EFN
10$:    TST       IOSB+2        ;ANY DATA IN BUFFER?
        BEQ       30$           ;IF EQ NO
        MOV       @CURPT,R0     ;GET NEXT VALUE FROM BUFFER
        DEC       IOSB+2        ;REDUCE NUMBER OF ENTRIES
        ADD       #2,CURPT      ;UPDATE BUFFER READ POINTER
        CMP       CURPT,LSTPT   ;END OF BUFFER?
        BLOS      20$           ;IF LOS NO
        MOV       FSTPT,CURPT   ;RESET BUFFER READ POINTER
20$:    Process data value      ;
        BR        10$           ;TRY AGAIN
30$:    TSTB      IOSB          ;REQUEST TERMINATED?
        BNE       40$           ;IF NE YES
        WTSE$S    #EFN          ;WAIT FOR EFN
        BR        5$            ;
40$:    Determine reason for termination
```

For IO.MDA and IO.MDO, this protocol differs slightly. The user task maintains a write pointer and the driver a read pointer. The entire buffer must be full when the request is executed.

12.6.4 Use of ADJLPS for Input and Output

The following FORTRAN example illustrates the proper protocol for using ADJLPS for synchronous input and output.

Synchronous input:

```
     DIMENSION IBF(1004),IERR(2),INTVL(2)
C
C INITIATE SYNCHRONOUS A/D SAMPLING,
C
       INTVL(1)=2
       INTVL(2)=5
       CALL RTS(IBF,1004,160,INTVL,IEFN,6,6,IERR,50,16,15)
C
C INITIALIZE HALF BUFFER INDEX
C
       INDX=4
C
C WAIT FOR HALF BUFFER OF DATA
C
 10    CALL WAITFR(IEFN)
C
C CLEAR EVENT FLAG
C
 15    CALL CLREF(IEFN)
C
C PROCESS HALF BUFFER OF DATA
C
       SUM=0
       DO 20 I=1,500
       SUM=SUM+CVSWG(IBF(I+INDX))
 20    CONTINUE
       AVERG=SUM/500
C
C FREE HALF BUFFER FOR MORE DATA
C
       CALL ADJLPS(IBF,500)
C
C ADJUST BUFFER INDEX
C
       INDX=INDX+500
       IF(INDX.GE.1004) INDX=4
C
C CHECK IF ANOTHER HALF BUFFER OF DATA IS AVAILABLE
C
       IF(IERR(2).GE.500 GO TO 15
       IF(IERR(1).NE.0) GO TO end of sampling
       GO TO 10
```

Synchronous output:

```
       DIMENSION IBF(1004),IERR(2),INTVL(2)
C
C FIRST BUFFER OF DATA MUST BE AVAILABLE AT START
C
C THIS EXAMPLE ASSUMES FIRST BUFFER IS FULL AT START
C
C START SYNCHRONOUS DIGITAL OUTPUT FUNCTION
C
       INTVL(1)=2
       INTVL(2)=5
       CALL SDO(IBF,1004,160,INTVL,IEFN,MASK,IERR,50,16,15)
C
C INITIALIZE HALF BUFFER INDEX
```

```
C
      INDX=4
C
C WAITFOR ROOM IN BUFFER
C
 10   CALL WAITFR(IEFN)
C
C CLEAR EVENT FLAG
C
 15   CALL CLREF(IEFN)
C
C CALCULATE VALUES TO PUT IN BUFFER
C
      X=(Y+2)*Z
      DO 20 I=1,500
      IBF(I+INDX)=X**5/A
 20   CONTINUE
C
C SIGNIFY ANOTHER HALF BUFFER IS FULL
C
      CALL ADJLPS(IBF,500)
C
C ADJUST BUFFER INDEX
C
      INDX=INDX+500
      IF(INDX.GE.1004) INDX=4
C
C CHECK IF ANOTHER HALF BUFFER IS EMPTY
C
      IF(IERR(2).GE.500) GO TO 15
      IF(IERR(1).NE.0) GO TO end of sampling
      GO TO 10
```

NOTE

In both the examples above, care is taken to ensure that the program stay "in sync" with the driver. If the program "loses" its position with respect to the driver the function must be stopped and restarted since this is the only way to recover. Caution should be exercised to ensure that the program sequence above be used to avoid a possible loss of data.

CHAPTER 13

PAPER TAPE READER/PUNCH DRIVERS

13.1 INTRCDUCTION

The RSX-11M paper tape reader/punch drivers support the PC11 paper tape reader/punch and the PR11 paper tape reader. The PC11 is a high-speed reader/punch capable of reading eight-hole, unoiled, perforated paper tape at 300 characters per second, and punching tape at 50 characters per second. The PR11 has the same characteristics as the paper tape reader portion of the PC11. All transfers are image mode only, with no interpretation of data.

13.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the GET LUN INFORMATION system directive (the first characteristics word) contains the following information for paper tape devices. A bit setting of 1 indicates that the described characteristic is true for these devices.

Bit	Setting	Meaning
0	1	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	0	Directory device
4	0	Single-directory device
5	0	Sequential device
6-12	0	Reserved
13	0	Device mountable as a communications channel
14	0	Device mountable as a FILES-11 volume
15	0	Device mountable

Words 3 and 4 of the buffer are undefined; word 5 indicates the default buffer size, which is 64 bytes for paper tape devices.

13.3 QIO MACRO

Table 13-1 lists the standard functions of the QIO macro that are valid for the paper tape reader/punch.

Table 13-1
Standard QIO Functions for the Paper Tape Reader/Punch

Format		Function
QIO$C	IO.ATT,...	Attach device
QIO$C	IO.DET,...	Detach device
QIO$C	IO.KIL,...	Cancel I/O requests
QIO$C	IO.RLB,...,<stadd,size>	Read logical block (reader only)
QIO$C	IO.RVB,...,<stadd,size>	Read virtual block (reader only)
QIO$C	IO.WLB,...,<stadd,size>	Write logical block (punch only)
QIO$C	IO.WVB,...,<stadd,size>	Write virtual block (punch only)

where: stadd is the starting address of the data buffer (may be on a byte boundary)

size is the data buffer size in bytes (must be greater than zero)

IO.KIL never cancels an in-progress read request. In-progress write requests are cancelled only when the punch driver is waiting for the punch to become ready at the start of a transfer.

The paper tape drivers support no device-specific functions.

13.4 STATUS RETURNS

Table 13-2 lists error and status conditions that are returned by the paper tape reader/punch drivers.

Table 13-2
Paper Tape Reader/Punch Status Returns

Code	Reason
IS.SUC	Successful completion. The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading or writing.
IS.PND	I/O request pending. The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with zeros.
IE.ABO	Operation aborted. The I/O request was cancelled while in progress or while still in the I/O queue.

Table 13-2 (Cont.)
Paper Tape Reader/Punch Status Returns

Code	Reason
IE.DAA	Device already attached. The physical device unit specified in an IO.ATT function was already attached by the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task.
IE.DNA	Device not attached. The physical device unit specified in an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.
IE.DNR	Device not ready. The reader and punch drivers return this code when a time-out occurs. The reader driver also returns this code when an error condition (see Section 13.4.1) is encountered before the initiation of the first transfer after an ATTACH command has been issued.
IE.EOF	End-of-file encountered. The reader driver encountered an error condition (see Section 13.4.1) at a time other than the initiation of the first read after a valid ATTACH command. The second word of the I/O status buffer contains a count of bytes successfully read before the error condition was encountered.
IE.IFC	Illegal function. An illegal function code was specified in an I/O request that is not legal for the respective paper tape drivers.
IE.OFL	Device off-line. The physical device unit associated with the LUN specified in the QIO directive was not on-line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.SPC	Illegal address space. The buffer specified for a read or write request was partially or totally outside the address space of the issuing task. Alternatively, a byte count of zero was specified.
IE.VER	Unrecoverable hardware error (punch only). The punch driver encountered an error condition (see Section 13.4.1) at a time other than the initiation of a transfer. Section 13.4.2 describes the action of the punch driver when an error condition is encountered upon the initiation of a transfer.

13.4.1 Error Conditions

There are four error conditions that are indistinguishable to the paper tape drivers. These conditions are:

1. No tape
2. Reader off-line
3. Power low
4. Hardware malfunction

13.4.2 Ready Recovery

When the punch driver encounters an error condition upon the initiation of a transfer, the following message is displayed:

```
*** PPn:  -- NOT READY
```

where n is the unit number of the paper tape punch that is not ready. This message is repeated every 15 seconds until the error condition is corrected, or until the I/O request is cancelled. When the error condition has been corrected, the transfer will begin within one second.

13.5 PROGRAMMING HINTS

This section contains information on important programming considerations relevant to users of the paper tape drivers described in this chapter.

13.5.1 Special Action Resulting from Attach and Detach

When an Attach or Detach is issued to the punch, the punch driver initiates a transfer of 170 (decimal) nulls. Upon the first read after an attach to the reader, all nulls preceding the first non-null character on the tape are read and discarded by the reader driver.

13.5.2 Reading Past End-of-Tape

When the reader driver reads past the physical end-of-tape, it normally generates at least two incorrect data bytes. These bytes are included in the byte count returned by the driver. User software that does not prevent reads past the physical end-of-tape should discard at least the last six characters in the buffer when IE.EOF is returned by the driver.

CHAPTER 14

INDUSTRIAL CONTROL LOCAL AND REMOTE SUBSYSTEMS

14.1 INTRODUCTION

ICS11 and ICR11 are local, and remote process I/O subsystems respectively. They operate under program control as devices capable of interrogating digital and analog input, and driving digital and analog output.

14.1.1 Hardware Configuration

A single ICS or ICR controller can handle up to 16 I/O modules in any configuration; a module contains 16 bits of input or output data, providing a total of 256 digital points. Up to 12 ICR or ICS units are supported. The ICS/ICR driver is tailored to the user's needs, interactively, through the SYSGEN (System Generation program) dialogue. The driver is capable of handling any combination of ICR or ICS controllers installed on a single system.

14.1.1.1 Address Assignments - Each ICR11A Unibus interface or ICS11 file box must be configured for individually addressable interrupt vectors, Control and Status Registers (ICSR), and module Address Registers (ICAR) as shown below.

Table 14-1
ICS/ICR Address Assignments

ICS/ICR UNIT NO.	MODULE ADDRESSES	ICSR/ICAR ADDR.	INTERRUPT VECT.
0	171000-171036	171770-171776	234-236
1	171040-171076	171760-171766	XXX-XXX+2
2	171100-171136	171750-171756	XXX+4-XXX+6
3	171140-171176	171740-171746	XXX+10-XXX+12
4	171200-171236	171730-171736	XXX+14-XXX+16

Table 14-1 (Cont.)
ICS/ICR Address Assignments

ICS/ICR UNIT NO.	MODULE ADDRESSES	ICSR/ICAR ADDR.	INTERRUPT VECT.
5	171240-171276	171720-171726	XXX+20-XXX+22
6	171300-171336	171710-171716	XXX+24-XXX+26
7	171340-171376	171700-171706	XXX+30-XXX+32
10	171400-171436	171670-171676	XXX+34-XXX+36
11	171440-171476	171660-171666	XXX+40-XXX+42
12	171500-171536	171650-171656	XXX+44-XXX+46
13	171540-171576	171640-171646	XXX+50-XXX+52

NOTES

nnnnn6 = Control and Status Register
nnnnn4 = Address Register

Additional controllers are assigned vector addresses above 300.

14.1.1.2 Supported I/O Modules - The following modules, all optional, are supported by the ICS/ICR driver.

D/A Converters

IDA-OA - 4-channel digital-to-analog converter.

A/D Converters

IAD-IA - 8-channel wide-range differential analog-to-digital converter.
IMX-IA - 16-channel flying capacitor relay multiplexer.

Counters

IDC-IC - 16-bit binary counter.

Bistable Digital Outputs

IDC-OA - D/C flip/flop driver.
IAC-OA - A/C flip/flop driver.
IRL-OA - Latching relay output.
IRL-OB - Flip/flop relay output.

Momentary Digital Output

IDC-OB - D/C single-shot driver.
IAC-OB - A/C single-shot driver.

Digital Inputs (Noninterrupting)*

IDC-IA - D/C voltage sense input.
IDC-ID - D/C voltage input module.
IAC-IA - A/C voltage input module.

Digital Inputs (Interrupting)

IDC-IB - D/C voltage interrupt input.
IAC-IB - A/C voltage interrupt input.

Terminal Input/Output

110 CPS Remote Terminal Interface to ICR11.

14.1.2 Alternate ICS11 Support

The ICS11 Industrial Control Subsystem is supported either by the UDC11 or ICS/ICR-11 device driver. If the system does not have an ICR11 controller, and if a driver of minimum size is required, then UDC11 support should be considered. The hardware requirements for such support are as follows:

1. Each file box must be assigned to the same interrupt vector address (normally 234).

2. The control and status register within each file box must appear at the same address within the I/O page (normally 171776).

If support of the IAD-IA A/D converter is required, the following module addressing and installation conventions are imposed:

1. Each IAD-IA converter and its associated IMX-IA relay multiplexer is assigned a fixed block of 120 logical channel numbers. No more than 32 IAD-IA converters may be installed in a single system. Based on this convention, A/D converter 0 occupies channels 0-119, A/D converter 1 occupies 120-239 etc.

2. Regardless of the actual number of IMX-IA multiplexers installed, each converter preempts a block of eight contiguous module slots.

3. The slots reserved for all A/D converters and multiplexers must occupy a block of contiguous module slots.

If necessary, the vector and address changes can be made by Field Service personnel. Assuming the hardware configuration is correct, the user can implement the desired UDC11 software support by answering all SYSGEN questions relating to the UDC11 in the affirmative.

If the additional ICS/ICR-11 driver features are required (at a commensurate increase in the memory requirements), then each ICS11

*Note that noninterrupting input modules are accessed directly by a task. Hence, while FORTRAN interface routines are available, no support for such modules is included in the driver.

file box must be configured for individually addressable interrupt vectors and control status registers. This change can be performed by Field Service personnel. The necessary software support is incorporated by answering all SYSGEN questions relating to the ICS/ICR-11 in the affirmative.

The additional ICS11 capabilities provided by the ICS/ICR-11 driver may be summarized as follows:

1. Multi-controller, parallel operation.
2. Increased A/D conversion throughput.
3. Activation of tasks directly from digital interrupts or counters.
4. No requirement to install modules of the same type in contiguous slots.

Section 14.7 summarizes the software differences between the UDC and ICS/ICR drivers in detail.

14.1.3 Software Support

ICS/ICR operations are divided into two categories:

1. Functions performed directly by any task.
2. Functions requiring driver services.

Direct functions are accomplished through memory references to the ICS/ICR registers on the I/O page. In a protected system any task may gain restricted access to the device registers by linking to a global common block that resides within the appropriate physical memory limits. Direct functions consist of:

1. Reading counter modules.
2. Reading any digital input module.

Driver requests are divided into the following categories:

1. Noninterrupting output functions
    a. Bistable (flip/flop) digital output
    b. Analog output
    c. Momentary (single-shot) digital output
2. Requests for interrupting functions
    a. Analog input
    b. Remote terminal output

3. Requests for unsolicited interrupts

    a. Digital interrupts

    b. Counter interrupts

    c. Remote terminal input

    d. Remote unit or serial line errors

With the exception of A/D input and remote terminal output, all functions are complete upon return to the user's task.

Under RSX-11M noninterrupting output functions are immediately submitted to the controller through a circular buffer that is filled at driver level and emptied at interrupt level. A QIO is considered successfully completed when the request is inserted in the circular buffer.

The following operations are in this category:

1. Bistable digital outputs

2. Analog outputs

3. Momentary digital outputs

Interrupting functions are those operations that generate an interrupt within some fixed time after initiation. The driver allows a list of multiple transactions to be specified in a single QIO. Each transaction is initiated in sequence without waiting for the preceding interrupt, until either the list is exhausted or all modules of the specified type are active. The following operations are in this category:

1. A/D inputs

2. Remote terminal output

Unsolicited interrupts may require no initiation by the processor and occur at indeterminate intervals. The following functions are in this category:

1. Interrupting digital inputs

2. Counter modules

3. Remote terminal input

4. Error interrupts

All unsolicited interrupt data, except for errors, may be placed in a task-provided circular buffer. On interrupt, an event flag specified by the task is set. Such data for each module type is supplied to only one task per controller. In addition, the driver will activate selected tasks on the occurrence of digital or terminal input interrupts.

Error interrupts are described further on in this chapter.

Terminal support is restricted to passing terminal data between the device and a task. The only special character is Control-C (003), which may cause a user-specified task to be made active. There is no other special processing for terminal I/O except that the parity bit is removed. This is similar to the terminal driver function of IO.RAL.

1. MCR is not invoked.
2. Characters are not echoed.
3. Carriage control is not performed.
4. TABs, RUBOUTs, etc. are not recognized.
5. Line terminators are not recognized.
6. Fill characters are not generated.

14.1.4 UDC11 Software Compatibility

Many of the MACRO and FORTRAN interfaces described in the following paragraphs are fully compatible with existing UDC11 applications software; however, the user should consult section 14.7 for a summary of differences that do exist between UDC and ICS/ICR software.

14.2 LUN INFORMATION

A request for logical unit information returns the following device-dependent data in words 2 through 5 of the buffer:

WD 02 - 0

WD 03 - undefined

WD 04 - undefined

WD 05 - 0

14.3 ASSEMBLY LANGUAGE INTERFACE

Table 14-2 summarizes standard and device-specific QIO functions supported by the ICS/ICR driver.

Table 14-2
Summary of ICS/ICR-11 QIO Functions

QIO$C IO.CCI,...<stadd,sizb,tevf>	Connect a buffer to digital interrupts
QIO$C IO.CTI,...<stadd,sizb,tevf,arv>	Connect a buffer to counter interrupts
QIO$C IO.CTY,...<stadd,sizb,tevf>	Connect a buffer to terminal interrupts

Table 14-2 (Cont.)
Summary of ICS/ICR-11 QIO Functions

QIO$C IO.DCI,...	Disconnect a buffer from digital interrupts
QIO$C IO.DTI,...	Disconnect a buffer from counter interrupts
QIO$C IO.DTY,...	Disconnect a buffer from terminal interrupts
QIO$C IO.FLN,...	Set controller offline
QIO$C IO.ITI,...<mn,ic>	Initialize a counter
QIO$C IO.LDI,...<tname,[tevf],pn,csm>	Link task to digital interrupts
QIO$C IO.LKE,...<tname,,[tevf]>	Link task to error interrupts
QIO$C IO.LTI,...<tname,,[tevf],cn,[arv]>	Link task to counter interrupts
QIO$C IO.LTY,...<tname,,[tevf]>	Link task to remote terminal interrupts
QIO$C IO.MLO,...<opn,pp,dp>	Open or close bistable digital output points
QIO$C IO.MSO,...<opn,dp>	Pulse single-shot digital output points
QIO$C IO.NLK,...<tname>	Unlink a task from all interrupts
QIO$C IO.NLN,...	Place ICS/ICR controller online
QIO$C IO.RAD,...<stadd>	Read activating data
QIO$C IO.RBC,...<stadd,size,stcnta>	Initiate multiple A/D conversions
QIO$C IO.SAO,...<chn,vout>	Perform analog output
QIO$C IO.UDI,...<tname>	Unlink a task from digital interrupts
QIO$C IO.UER,...<tname>	Unlink a task from error interrupts
QIO$C IO.UTI,...<tname>	Unlink a task from counter interrupts
QIO$C IO.UTY,...<tname>	Unlink a task from terminal interrupts.
QIO$C IO.WLB,...<stadd,sizb>	Transmit data to the ICR Remote terminal

Where:

arv	is the starting address of a buffer containing initial or reset counter values. The buffer must be aligned on a word boundary.
chn	is the D/A channel number
cn	is the counter number
csm	is the change-of-state mask
dp	is the binary data pattern
ic	is the initial count
mn	is the module number
opn	is the first latching digital output point number. This value must be on a module boundary (evenly divisible by 16)
sizb	is the data buffer size in bytes. For a circular buffer connected to unsolicited interrupts, this value must be even and large enough to include one entry plus the 2-word header
size	is the data and control buffer size in bytes. This value must be an even number that is greater than zero.
stadd	is the starting address of the data buffer (must be on a word boundary)
staddb	is the starting address of the terminal output buffer. May be aligned on a byte boundary
stcnta	is the starting address of the control buffer (must be on a word boundary); each control buffer word must be constructed as described in paragraph 14.3.2
tevf	is an event flag number in the range 1 to 64
tname	is a 2-word task name composed of 1 to 6 alphanumeric characters in RADIX-50 format
vout	is a binary number between 0 and 1023. that is to be converted to an analog output

The following sections contain a detailed description of each function. In the discussion of QIO request parameters, the following conventions apply.

All numbering is relative.

Module numbers start at 0 beginning with the first module of a given type. Increasing module numbers correspond to increasing physical bus addresses.

Channel numbers start at 0, with channel 0 as the first channel on the first module of a given type.

Point numbers start at 0 with point 0 as the first point on the first module of a given type. Points within a module are numbered "from right to left" in increasing order.

It should be remembered that there is no requirement for modules of a given type to occupy contiguous slots; thus, for example, digital points 15(10) and 16(10) need not reside on physically adjacent modules.

It is assumed that the number of points or channels per module is a constant for each generic type. Specifically the following weights apply:

1. Each Digital I/O Module contains 16 points.

2. Each Counter Module contains 1 channel.

3. Each D/A Module contains 4 channels.

4. Each A/D Converter contains 120 channels.

As stated above, an A/D converter is assigned a block of 120 channels. The number of channels in use within the block depends on the number of multiplexers installed. The driver will reject an attempt to address a nonexistent channel.

The table below illustrates the relationship between physical slot numbers, bus addresses and relative addresses for a given configuration. It is referred to in the following discussion.

As noted, a block of 120 relative addresses is reserved for each A/D converter. The converter and multiplexer in slots 10 and 11 contain channels 0 through 24. The converter in slot 17 contains channels 120 through 127. An attempt to access a nonexistent channel (e.g., channel 30 or channel 129) will be rejected by the driver.

The user should observe that the flip-flop drivers in slots 13 and 15 contain relative point numbers 0 through 15, and 16 through 29 although the modules are not physically adjacent. In general, the relationship between slot number, module type, bus address, and relative address is as follows:

1. A set of contiguous relative addresses is defined for each module of a given type that is installed. Each relative address, when qualified by type, uniquely identifies a digital point or channel.

2. A set of slot numbers and bus addresses, possibly not contiguous, is occupied by all modules of a given type. Such addresses may be assigned solely on the basis of hardware and installation considerations. Increasing relative addresses correspond to increasing bus addresses.

Table 14-3
Sample ICS/ICR Configuration

Unit: 0

Slot Number	Type	Bus Address	Relative Addresses
9.	Counter	171000	0
10.	A/D Converter	171002	0-119.
11.	A/D Multiplexer	------	-----
12.	Counter	171006	1.
13.	Flip Flop driver	171010	0-15.
14.	D/A Converter	171012	0-3
15.	Flip Flop driver	171014	16.-31.
16.	D/A Converter	171016	4-7.
17.	A/D Converter	171020	120.-239.

14.3.1 General Error Status Returns

The following error status returns apply uniformly to all requests:

IE.ABO - Operation aborted. The specified operation was cancelled via IO.KIL or the request timed out while the unit was offline.

IE.OFL - Controller offline. The physical device unit associated with the LUN specified in the QIO directive was not online. An ICS/ICR controller may be offline because a device check during bootstrap load has indicated that the controller is not in the configuration.

IE.DNR - Controller not ready. A nonrecoverable controller error has been detected.

IE.IFC - Illegal function. A function code was included in an I/O request that is illegal for the ICS/ICR. The function may also refer to an ICS/ICR module type or function that was not specified during system generation.

14.3.2 A/D Input - Read Multiple A/D Channels

This function provides the capability of reading several A/D channels at any permissible gain. The driver is capable of initiating parallel transfers when more than one A/D converter is installed in a file box; however, only one interrupt module request (remote terminal or A/D) may be in progress at a given time.

QIO DPB format:

```
QIO$C IO.RBC,...<stadd,size,stcnta>
```

where:

stadd = the starting address of the data buffer (must be on a word boundary).

size = the data buffer size in bytes (must be even and greater than zero); the control buffer is the same size.

stcnta = the starting address of the control buffer (must be on a word boundary); each control buffer word must be constructed as shown in Table 14-6.

Return Status:

IS.SUC - Function successfully completed.

IE.BAD - Illegal channel or gain code specified.

IE.BYT - Data buffer is byte aligned. Alternatively, the length of the buffer is not an even number of bytes.

IE.DNR - Device not ready. A/D converter interrupt timeout occurred.

Note that the second I/O Status word contains a count of the number of conversions successfully completed.

One control word is paired with each data word. That is, the data appearing in a data array element is obtained using the gain and channel number specified in the corresponding element of the control array. Control words specify the gain and channel in the following format:

Bits	Meaning				
0-11	Channel Number range: 0-1919				
12-15	Gain value for this sample. The binary value is as follows:				GAIN
	15	14	13	12	IAD-IA
	0	0	0	0	1
	0	0	0	1	2
	0	0	1	0	ILLEGAL
	0	0	1	1	ILLEGAL
	0	1	0	0	10
	0	1	0	1	20
	0	1	1	0	ILLEGAL
	0	1	1	1	ILLEGAL
	1	0	0	0	50
	1	0	0	1	100
	1	0	1	0	ILLEGAL
	1	0	1	1	ILLEGAL
	1	1	0	0	200
	1	1	0	1	1000
	1	1	1	0	ILLEGAL
	1	1	1	1	ILLEGAL

Upon receipt and validation of the parameters within the I/O packet, the driver will initiate the following sampling procedure:

1. The control word is fetched and tested for validity (i.e., for legal gain and channel). If an error is encountered or no further control words remain, processing is terminated as described in Step 4.

2. Assuming the A/D converter board is idle, the driver starts the conversion, sets this resource busy and returns to Step 1. If the converter is busy the driver returns control to the system after saving the data required to initiate the conversion when the channel becomes idle.

3. On the occurrence of an A/D interrupt, the interrupt service routine initiates the appropriate processing at the non-interrupt level that will either set the channel idle or initiate a previous request stored during Step 2. The occurrence of the latter results in processing of additional control words as described in Step 1.

4. A/D requests are terminated under any of the following conditions:

   a. All control words have been processed.

   b. A hardware error has occurred.

   c. An error in a control word has been detected.

Regardless of the cause, the driver cannot complete request processing until all pending A/D transfers have gone to completion.

Because of overlapped processing, multiple errors can occur (e.g., a hardware error and an erroneous control word). The driver will return the status associated with the earliest transaction that caused an error condition. Thus, at the user interface, the driver will appear to execute all conversions sequentially.

14.3.3 Analog Output

This function provides the capability of setting a single analog output channel to a specified voltage.

QIO DPB format:

```
QIO$C IO.SAO,...<chn,vout>
```

where:

chn = the output channel number

vout = the output voltage representation

Output voltage varies linearly with the binary input to the channel, where 0 to plus ten volts (+10v.) is represented by integers from 0 to 1023.

Return Status:

IS.SUC - Function submitted for output to controller.

IE.MOD - Nonexistent D/A channel was specified.

The second I/O status word is zero.

14.3.4 Single-Shot Digital Output - Multi-Point

This function provides the capability of pulsing a field of up to 16 momentary digital output points. Fields must be aligned on module boundaries.

QIO DPB format:

```
QIO$C IO.MSO,...<opn,dp>
```

where:

opn = the starting digital output point number. Point number must be aligned on a module boundary (i.e., must be a multiple of 16).

dp = the 16-bit mask. One point is pulsed corresponding to each bit set in the mask word.

Return Status:

IS.SUC - Function submitted for output to the controller.

IE.MOD - Invalid starting point number specified. Point is nonexistent or not aligned on a module boundary.

14.3.5 Bistable Digital Output - Multi-Point

This function provides the capability of setting or resetting a field of up to 16 bistable digital output points. Fields must be aligned on a module boundary.

QIO DPB format:

```
QIO$C IO.MLO,...<opn,pp,dp>
```

where:

opn = the starting digital output point number.

pp = the 16-bit mask.

dp = the data pattern.

A bit is set in the mask word for each point that may change state. The state of points corresponding to reset mask bits is unaltered.

When the mask bit is set, the output will be "closed" if the data bit is set and "open" if the data bit is clear.

Return Status:

IS.SUC - Function submitted for output to the controller.

IE.MOD - Invalid starting point number specified. Point does not exist or is not aligned on a module boundary.

14.3.6 Unsolicited Interrupt Processing

Unsolicited interrupts consist of the following:

1. Digital interrupts
2. Counter interrupts
3. Remote terminal input
4. Hardware Errors

Based on the type of interrupt, the driver may dispose of the interrupt data in one or more of the following ways:

1. The data may be furnished to a task that has issued a request to continually monitor such information.
2. A task may be activated by a specific input. That is, a dormant task can be requested to run, or an event flag may be set if the task is currently active.

The driver will allow continual monitoring for digital, counter, and terminal inputs with the provision that, for each controller, only one task per module type may receive such inputs.

Task activation is permitted for digital, terminal, and error interrupts. The processing related to hardware errors is discussed in section 14.5. Activation of tasks by digital counter and terminal inputs is covered in section 14.3.7.

The driver functions described in the following paragraphs allow a task to continually receive interrupt data. To monitor such data a task must provide:

1. A buffer that is filled by the driver and emptied by the task in circular fashion.
2. An event flag that will be set upon the occurrence of each interrupt.

The driver will connect a single task per controller to receive interrupts from a specific module type.

The buffer to be connected has the format shown below:

FORTRAN Index	Contents
1	driver index
2	user index
3	entry
4	entry
.	.
.	.
.	.

The buffer consists of a two-word header containing the driver and user index, as shown, followed by a data area that is subdivided into fixed length entries. Each entry consists of an entry existence indicator followed by one or more words of device-dependent data. Such information usually consists of module data, relative module number, and a code identifying a module type. On the occurrence of an interrupt, the driver enters data in the location currently indicated by the driver index. This index can be considered as a FORTRAN index into the buffer. That is, the first location in the buffer is associated with the index 1. The beginning of the data area is associated with the first entry, index 3. Entries are made in a circular fashion starting at the beginning of the data area, filling in order of increasing memory address, and wrapping around to the beginning of the data area when there is insufficient space for an entry at the end. Note that the size of the data area must be an integer multiple of the entry size.

It is expected that the connected task will maintain the user index, ensuring that it indicate where, in the buffer, the task is to process interrupt data next.

When the task is activated by the driver, it should process data in the buffer starting at the location indicated by its pointer, and continuing in circular fashion until an existence indicator is encountered that is zero.

The existence indicator is set to +1 when a buffer entry is made. Except to record a hardware error, the contents of an entry are not altered by the driver if the indicator is nonzero. Hence, when a requester has removed or processed the entry, he must clear the existence indicator in order to free the buffer entry position. If the driver detects a nonzero indicator, i.e., data input has occurred in a burst sufficient to overrun the buffer, the data is discarded and a count of data overruns is incremented. The count is maintained in the entry existence indicator which, as noted above, is set to +1 to indicate no overruns between entries, +2 to indicate a hardware error entry, or a negative value recording the two's complement of the number of times data has been discarded between entries. The overrun count will never be allowed to wrap around to a positive value.

In the event of a nonrecoverable controller error (remote unit power-fail or hard data error) all connected tasks are activated with the following entry in the circular buffer:

```
WD 00           Hardware error indicator (+2)
     .
     .
     .
WD nn           Contents of ICSR register
WD nn+1         Physical unit number
WD nn+2         Generic code indicator
                set to 177770(8)
```

nn = offset to module data word.

This entry is always placed in the buffer regardless of overflow status.

The error flags are obtained from the controller ICSR word at the time the error was detected (see Table 14-7).

14.3.6.1 Connect to Digital Interrupts - This function allows a single task to receive digital interrupt data.

QIO DPB format:

```
QIO IO.CCI,...<stadd,sizb,tevf>
```

where:

stadd = starting address of buffer to be connected (must be word aligned)

sizb = length of buffer in bytes (must be even). Minimum buffer length is 14 bytes

tevf = trigger event flag number

Status:

IS.SUC - Function successfully completed. Second I/O status word contains the number of words passed per interrupt in the low byte, and the initial FORTRAN index in the high byte.

IE.BYT - Buffer address is byte aligned or length is an odd number of bytes.

IE.CON - Interrupt already connected to another task.

IE.IEF - Illegal event flag was specified i.e., not in the range 1-64.

IE.PRI - Task checkpointable and not fixed in memory.

IE.SPC - Interrupt circular buffer was not wholly within the address space of the task. Alternatively, the buffer was too small for a single data entry (7 words minimum).

Entry Format:

WD 00 - Existence Indicator
WD 01 - Change of state indicator
WD 02 - Module data
WD 03 - Relative module number
WD 04 - Generic Code 1, 2 or 3, indicating a digital interrupt

The contents of the existence indicator have been described previously.

The change-of-state indicator records those bits for which a change of state in the direction of interest has been detected. The direction of the change may be from 0 to 1 (point closed (PCL)) or 1 to 0 (point open (POP)) depending upon the PCL or POP jumper connections on the digital interrupt module. The driver will assume that at least one of these signals is always asserted.

The relative module number indicates the module on which the change of state was recognized.

The module data word records data received at the time the interrupt was serviced.

The generic code identifies the type of module that caused the interrupt. A digital interrupting module may have the value 1, 2 or 3 as selected by user-installed jumpers on the module.

14.3.6.2 Disconnect from Digital Interrupts - This function allows a task to terminate the processing of digital interrupt data.

QIO DPB format:

```
QIO$C IO.DCI,...
```

Return Status:

IS.SUC - Function successfully completed. Second I/O status word is zero.

IE.CON - Task was not connected. Second I/O status word is zero.

14.3.6.3 Connect to Counter Module Interrupts - This function allows a single task to receive counter interrupt data.

QIO DPB format:

```
QIO$C IO.CTI,...<stadd,sizb,tevf,arv>
```

where:

stadd = starting address of circular buffer (must be word aligned)

sizb = length of buffer in bytes (must be even). Minimum buffer length is 12 bytes.

tevf = trigger event flag number

arv = starting address of table of initial counter values (must be word aligned)

Word 03 defines an array of initial counter values. One entry is required for each counter installed in a physical unit. Entries are paired with modules in logically ascending sequence. The counter is set to the initial value upon receipt of the connect function and whenever an overflow interrupt occurs (i.e., when the count reaches zero).

Return Status:

IS.SUC - Function successfully completed. The second I/O status word contains the number of words passed per interrupt in the low byte, and the initial FORTRAN index in the high byte.

IE.BYT - Buffer address is byte aligned or length is an odd number of bytes.

IE.CON - Interrupt already connected to another task.

IE.IEF - Illegal event flag was specified, i.e., not in the range 1-64.

IE.PRI - Task checkpointable and not fixed in memory.

IE.SPC - Interrupt circular buffer or table of initial values was not wholely within the address space of the task. Alternately, the buffer was too small for a single data entry (6 words minimum).

Entry Format:

WD 00 - Existence indicator

WD 01 - Module data

WD 02 - Relative module number

WD 03 - Generic code (4, 5 or 6)

14.3.6.4 Set Counter Initial Value - This function allows a counter initial value to be established. A task need not be connected to counter interrupts to perform this function.

QIO DPB format:

```
QIO$C IO.ITI,...<mn,ic>
```

where:

mn = relative module number

ic = new initial count

Return Status:

IS.SUC - New value submitted for output to the controller. The second word of I/O status is set to zero.

IE.MOD - Nonexistent module number specified.

Upon receipt of the request, the new initial value is immediately queued for output to the controller. The counter will be reinitialized with this value on overflow if a task is connected to counter interrupts.

14.3.6.5 Disconnect from Counter Interrupts - This function allows a task to terminate counter interrupt processing.

QIO DPB format:

```
QIO$C IO.DTI,...
```

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is set to zero.

IE.CON - Task was not connected to timer interrupts.

After disconnect is complete, counters will not be reset to the initial value at the time of the interrupt.

14.3.6.6 Connect to Terminal Interrupts - This function allows a task to receive terminal inputs from the selected ICR11 controller.

QIO DPB format:

```
QIO$C IO.CTY,...<stadd,sizb,tevf>
```

where:

stadd = address of circular buffer (must be word aligned)

sizb = length of buffer (must be even). Minimum buffer length is 12 bytes.

tevf = trigger event flag number

Return Status:

IS.SUC - Function successfully completed. The second I/O status word contains the number of words passed per interrupt in the low byte, and the initial FORTRAN index in the high byte.

IE.BYT - Buffer is byte aligned or length is an odd number of bytes

IE.CON - Interrupt already connected to another task.

IE.IEF - Illegal event flag was specified, i.e., not in the range 1 to 64.

IE.MOD - Nonexistent device. Controller is ICS11.

IE.SPC - Interrupt circular buffer was not wholly within the address space of the task. Alternatively, the buffer was too small for a single entry (6 words minimum).

Entry Format:

WD 00 - Existence indicator

WD 01 - High byte = 0, low byte = terminal input character

WD 02 - Relative module number (normally 0)

WD 03 - Generic code indicator (normally 0)

Note that words 2 and 3 are nonzero only when the entry was made as the result of a nonrecoverable controller error.

All remote terminal data is conveyed to the requesting task as input, but with the parity bit removed.

NOTE

Remote terminal input is not echoed by the driver.

14.3.6.7 Disconnect from Terminal Input - This function allows a task to discontinue the processing of terminal input.

QIO DPB format:

QIO$C IO.DTY,...

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is set to zero.

IE.CON - Task was not connected to remote terminal interrupts.

14.3.7 Activating a Task by Unsolicited Interrupts

The functions described in the following paragraphs provide the capability of:

1. Activating a task in response to unsolicited interrupts.
2. Interrogating the driver to determine the reason for activation.
3. Removing a task from the activation list.

The QIO DPB parameters specify the task name, an optional trigger event flag to be set if the task is active, and device-dependent parameters that identify the interrupt source. A task is linked to interrupts (i.e., made eligible for activation) provided that:

1. the resource exists,
2. the task is installed, and
3. no other task is linked to the resource.

If another task is linked to the resource, the driver will reject the request with a status of resource-in-use (IE.RSU). A resource is defined as a single interrupt point, remote terminal (Control-C input only), or counter module.

On the occurrence of the appropriate interrupt, the task is made active if dormant; otherwise, a trigger event flag, if specified, is set. The task may interrogate the driver to determine the conditions that caused activation, and to signify interrupt recognition. The function of the event flag is to allow such a task to recognize an event that has occurred while the task was active. Recognition is ensured prior to the completion of task execution by issuing the Exit If system directive followed by the Clear Event Flag directive.

The linkage between a task and a specific interrupt is removed by issuing the appropriate unlink request via the QIO directive.

Only one task may be associated with each interrupt source (i.e., one task per digital interrupt point, terminal input, or counter module.

NOTE

The MCR command REMOVE automatically unlinks a task from all interrupts.

14.3.7.1 Link a Task to Digital Interrupts - The following function allows a task to be activated on the occurrence of digital interrupts.

QIO DPB format:

```
QIO$C IO.LDI,...<tname,,[tevf],pn,csm>
```

where:

tname = a 1- to 6-character alphanumeric task name in 2-word, Radix-50 format

tevf = trigger event flag (0=none)

pn = point number (must be aligned on a module boundary)

csm = change-of-state mask

The change-of-state mask indicates those bits for which a change of state in the direction specified by the PCL and POP jumpers causes the task to be activated. Only one task may be linked to a given interrupt point. A zero change of state mask is not permitted.

Return Status

IS.SUC - Function successfully completed. The second word of I/O status is set to zero.

IE.BAD - Change-of-state mask set to zero.

IE.IEF - Trigger event flag not in the range 0-64.

IE.MOD - Nonexistent module or point. Not aligned on a module boundary.

IE.NOD - Insufficient dynamic memory to allocate secondary control block.

IE.NST - Task "tname" is not installed.

IE.RSU - One or more of the specified points is in use by other tasks.

14.3.7.2 Link a Task to Counter Interrupts - This function allows a task to be activated by means of an interrupt from a single counter module.

QIO DPB format:

```
QIO$C IO.LTI,...<tname,,[tevf],cn,[ic]>
```

where:

tname = a 1- to 6-character alphanumeric task name in 2-word Radix-50 format.

tevf = trigger event flag (0=none)

cn = relative module number

ic = counter value (optional)

The counter value if nonzero, is used to reinitialize the module in a manner similar to that described for the Set Counter function in section 14.3.6.4. Initialization may be bypassed by setting this parameter to zero.

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is set to zero.

IE.IEF - Trigger event flag parameter not in the range 0-64.

IE.MOD - Nonexistent module specified.

IE.NOD - Insufficient dynamic memory to allocate a secondary control block.

IE.RSU - Counter is linked to another task.

14.3.7.3 Link a Task to Terminal Interrupts - This function allows a task to be activated by means of an interrupt from a terminal module. The task will be activated only in response to the Control-C character (octal 003).

QIO DPB format:

```
QIO$C IO.LTY,...<tname,,[tevf]>
```

where:

tname = a 1- to 6-character alphanumeric task name in 2-word, Radix-50 format.

tevf = trigger event flag (0=none).

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is zero.

IE.IEF - Trigger event flag parameter not in the range 0-64.

IE.MOD - Nonexistent module (unit is ICS11 controller).

IE.NOD - Insufficient dynamic storage to allocate secondary control block.

IE.NST - Task "tname" is not installed.

IE.RSU - Remote terminal is linked to another task.

14.3.7.4 Link a Task to Error Interrupts - This function allows a single task to be activated whenever a remote unit power-fail or nonrecoverable serial line error is detected on any or all remote units in a system. Only one task within a system may be linked to error interrupts. Once linked, the selected task may receive error reports from any ICR controller.

QIO DPB format:

```
QIO$C IO.LKE,...<tname,,[tevf]>
```

where:

tname = a 1- to 6-character alphanumeric task name in 2-word Radix-50 format.

tevf = trigger event flag (0 = none)

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is zero.

IE.IEF - Trigger event flag parameter not in the range 0-64.

IE.IFC - No ICR11 subsystems are installed.

IE.NOD - Insufficient dynamic storage to allocate secondary control block.

IE.NST - Task "tname" is not installed.

IE.RSU - Another task is linked to error interrupts.

14.3.7.5 Read Activating Data - This function allows a task to determine the conditions that caused it to be activated.

QIO DPB format:

```
QIO$C IO.RAD,...<stadd>
```

where:

stadd = address of 6-word buffer to receive activation data (must be word aligned).

The buffer receives data in the following format:

WD 00 - Activation indicator

WD 01 - Physical unit number

WD 02 - Generic Code

WD 03 - Relative module number

WD 04 - Hardware dependent data

WD 05 - Hardware dependent data

The activation indicator is similar in function to the existence indicator used when reading circular buffer entries. The indicator is set to +1 on the occurrence of an interrupt to which the requesting task is linked, and the appropriate data is stored. The indicator is cleared when the data is solicited by the task. If an interrupt linked to the task occurs and the parameter is nonzero then the previously stored data is not modified and the driver sets this element with the two's complement of the number of linked interrupts not recorded.

The physical unit number specifies the controller that received the interrupt.

The generic code is identical to that specified for circular buffer entries, namely:

0 - Terminal (Control-C)

1,2,3 - Digital interrupt

4,5,6 - Counter interrupt

177770 - Fatal controller error

Hardware-dependent data is associated with generic code and will consist of the following:

Terminal:

WD 04 - Terminal buffer contents (low byte)

WD 05 - undefined

Digital Interrupts:

WD 04 - Module data

WD 05 - Change-of-state indicator

Counter:

WD 04 - Module data

WD 05 - undefined

Fatal Controller Error:

WD04 - Contents of ICSR register (see Table 14-7)

WD05 - Contents of ICAR register (see Table 14-8)

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is zero.

IE.BYT - Buffer address is aligned on an odd byte boundary.

IE.NLK - Task "tname" was not linked to interrupts.

IE.SPC - Buffer not totally within the task's address space.

14.3.7.6 Unlink a Task from Interrupts - The functions described in the following paragraphs provide the capability of:

1. Unlinking a task from all interrupts on a controller,

2. Selectively unlinking a task from interrupts by module type.

a. Unlink a Task from All Interrupts

This function unlinks a task from all interrupts on a given controller and from error interrupts.

QIO DPB format:

QIO$C IO.NLK,...<tname>

where:

tname = 1- to 6-character alphanumeric task name in 2-word Radix-50 format.

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is zero.

IE.NLK - Task "tname" was not linked to interrupts.

b. Unlink a Task from all Digital Interrupts

This function provides the capability of unlinking a task from all digital interrupt points on a controller.

QIO DPB format:

```
QIO$C IO.UDI,...<tname>
```

where:

tname = a 1- to 6-character alphanumeric task name in 2-word Radix-50 format.

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is zero.

IE.NLK - Task "tname" was not linked to the specified class of interrupt.

IE.NST - Task not installed.

IE.MOD - Nonexistent module type specified.

c. Unlink a Task from Counter Interrupts

This function provides the capability of unlinking a task from all counter module interrupts.

QIO DPB format:

```
QIO$C IO.UTI,...<tname>
```

where:

tname = a 1- to 6-character alphanumeric task name in 2-word Radix-50 format.

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is zero.

IE.NLK - Task "tname" was not linked to the specified interrupts.

IE.NST - Task not installed.

IE.MOD - Nonexistent module type specified.

d. Unlink a Task from Terminal Interrupts

This function provides the capability of unlinking a task from terminal interrupts.

QIO DPB format:

```
QIO$C IO.UTY,...<tname>
```

where:

tname = a 1- to 6-character alphanumeric task name in 2-word Radix-50 format.

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is zero.

IE.NLK - Task "tname" was not linked to the specified interrupts.

IE.NST - Task not installed.

IE.MOD - Nonexistent module specified (i.e., device is an ICS11 controller).

e. Unlink a Task from Error Interrupts

This function provides the capability of unlinking a task from all error interrupts.

QIO DPB format:

```
QIO$C IO.UER,...<tname>
```

where:

tname = 1- to 6-character alphanumeric task name in 2-word Radix-50 format.

Return Status:

IS.SUC - Function successfully completed. The second word of I/O status is zero.

IE.IFC - No ICR11 controllers exist in the system.

IE.NLK - Task "tname" was not linked to error interrupts.

IE.NST - Task not installed.

14.3.8 Terminal Output

This function allows a task to perform output to the terminal device. Characters are output exactly as they appear in the buffer. The carriage control parameter is not recognized. It should be noted that only one interrupt module request per controller (terminal or A/D) may be in progress at a given time. Thus, the driver will not initiate an A/D operation on a given controller, until any terminal output in progress for that controller has been completed.

QIO DPB format:

```
QIO$C IO.WLB,...<staddb,sizb>
```

where:

staddb = Buffer address (may be odd)

sizb = Byte count (may be odd)

Return Status:

IS.SUC - Function successfully completed. Second word of I/O status contains the number of bytes output.

IE.MOD - Nonexistent hardware function. Request was issued for an ICS11 controller.

14.3.9 Maintenance Functions

The functions described below allow a privileged task to enable and disable error reporting while troubleshooting or maintenance on a remote unit is in progress.

14.3.9.1 Disable Hardware Error Reporting - This function allows a privileged task to disable error reporting and error interrupts, and restrict access to the controller while remote unit troubleshooting or module calibration is in progress (see section 14.5.1). Upon receipt and validation of the request, error interrupts are disabled and subsequent controller timeouts are ignored. The occurrence of device timeout while A/D conversion or remote terminal input is in progress results in termination of the request with the error code IE.ABO.

When error reporting is disabled in this manner, access to the controller for input or output to I/O modules is restricted to privileged tasks. All other requests not requiring the transmission of data to or from the device, are permitted for all tasks. Such requests are as follows:

a. Disconnect from digital, counter, or remote terminal interrupts

b. Unlink from interrupts

c. Read activating data

d. Link to digital, remote terminal, or error interrupts

e. Connect a buffer to digital or remote terminal interrupts

All other requests not issued by a privileged task are rejected with the error code IE.DNR.

QIO DPB format:

```
QIO$C IO.FLN,...
```

Return Status:

IS.SUC - Function successfully completed.

IE.FLN - Unit already offline.

IE.PRI - Task not privileged.

14.3.9.2 Enable Hardware Error Reporting - This function allows a privileged task to enable error reporting and device error interrupts. Upon receipt and validation of the function, all device error interrupts are enabled and the unit is marked online. These actions are performed regardless of the current state of the unit.

QIO DPB format:

```
QIO$C IO.NLN,...
```

Return Status:

IS.SUC - Function successfully completed.

IE.PRI - Task not privileged.

14.3.10 Special Functions

14.3.10.1 I/O Rundown - An I/O rundown request from the Executive will automatically cause the task to be disconnected from all interrupts. The rundown operation is not finished until any A/D input in progress for the task, has been completed.

14.3.10.2 Kill I/O - The kill I/O function allows a task to initiate I/O rundown processing for itself on any device. Request processing is identical to that described for I/O rundown.

QIO DPB format:

```
QIO$C IO.KIL,...
```

Return Status:

IS.SUC - Function successfully completed.

14.4 FORTRAN INTERFACE

The following table lists the FORTRAN interface subroutines supported for the ICS/ICR subsystem. (D) indicates a direct access call.

Unless specifically noted, all subroutines are reentrant (but not necessarily position-independent) and may be placed in an absolute resident library.

Table 14-4
FORTRAN Interface

Subroutine	Function
AIRD/AIRDW	Input analog data from multiple channels in random sequence.
AISQ/AISQW	Read a series of sequential analog input channels at random gain.
AO/AOW	Perform analog output on several channels.
ASICLN/ ASUDLN	Assign a LUN to an ICS/ICR controller.
CTDI	Connect a circular buffer to receive digital interrupt data.
CTTI	Connect a circular buffer to receive counter interrupt data.
CTTY	Connect a circular buffer to receive ICR11 remote terminal data.
DFDI	Disconnect a buffer from digital interrupts.
DFTI	Disconnect a buffer from counter interrupts.
DFTY	Disconnect a buffer from remote terminal interrupts.
DI/DIW	Read several 16-point digital sense fields (D).
DOL/DOLW	Latch or unlatch several 16-point bistable output fields.
DOM/DOMW	Pulse multiple 16-point momentary digital output fields.
LNK	Link a task to unsolicited interrupts.
OFLIN	Suppress error reporting. Place unit in not ready status.
ONLIN	Enable error reporting. Return unit to ready status.
RCIPT	Read a single digital interrupt point (D).
RDACT	Read interrupt activation data.
RDDI	Read the digital interrupt circular buffer.
RDTI	Read the counter interrupt circular buffer.
RDCS	Read digital interrupt circular buffer. Return data on only those points for which a change of state has been recognized.

Table 14-4 (Cont.)
FORTRAN Interface

Subroutine	Function
RDWD	Read digital interrupt circular buffer. Return a full data word.
RSTI	Read a single counter module (D).
RTO/RTOW	Perform output to a remote ICR11 terminal.
UNLNK	Unlink a task from unsolicited interrupts.

14.4.1 Synchronous and Asynchronous Process Control I/O

The Instrument Society of America (ISA) standard provides for synchronous and asynchronous I/O. Synchronous I/O is indicated by appending a W to the name of the subroutine (e.g., AO/AOW). Except for analog input and terminal output, all QIOs issued by the process control subroutines are serviced immediately by the driver and are complete upon return to the issuing task. In such cases there is no functional difference between the synchronous and asynchronous forms; however, both forms of the name are recognized. In the case of A/D input and terminal output, the subroutines are functionally distinct. If the asynchronous form is used, execution continues and the calling program must periodically test the status word for completion.

14.4.2 Return Status Reporting

The I/O status parameter is a 2-word integer array. The first element of the array receives the status of the FORTRAN call in accordance with ISA convention.

This array serves two purposes:

1. It is the 2-word I/O status block to which the driver returns an I/O status code on completion of an I/O request.

2. The first word of the status block receives a status code from the FORTRAN interface subroutine in ISA-compatible format, with the exception of the I/O pending condition, which is indicated by a status of zero. The ISA standard code for this condition is +2.

For asynchronous analog input and terminal output, status is set by means of an asynchronous trap, therefore the trap mechanism must be enabled while these functions are in progress.

For compatibility, the two-word status block is also required for status returned by the direct access calls. Errors of this type that may be returned are:

Word 1 = 3	Number of points requested is zero.
Word 1 = +321	Invalid ICS/ICR module.

The status code must be interpreted in the context of the function requested; however, the following general conditions will apply:

Contents of Status Word 1	Meaning
0	Operation pending, I/O in progress
+1	Successful completion
+3	Error in a calling argument has been detected by the interface subroutine
3<Word 1< 300.	QIO directive rejected. Actual error code = -(WORD 1 - 3)
Word 1 > 300	Request rejected by driver. Actual error code = -(WORD 1 - 300)

Table 14-5 lists all possible status values: the FORTRAN value, assembly language mnemonic, actual value and related definition.

Table 14-5
Return Status Summary

FORTRAN INTERFACE VALUE	ASSEMBLY LANGUAGE VALUE	ASSEMBLY LANGUAGE MNEMONIC	DEFINITION
+0	+0	IS.PND	Operation pending.
+1	+1	IS.SUC	Successful completion.
+3	none	none	Error detected in FORTRAN calling sequence.
+4	-1	IE.UPN	Insufficient dynamic storage to allocate I/O packet.
+8	-5	IE.ULN	Unassigned LUN.
-6	-6	IE.LNL	LUN usage interlocked.
+99	-96	IE.ILU	Invalid LUN.
+100	-97	IE.IEF	Invalid event flag number.
+101	-98	IE.ADP	Part of DPB out of user's addressing space.
+102	-99	IE.SDP	Invalid DIC or DPB size.
+301	-1	IE.BAD	Bad parameters.
+302	-2	IE.IFC	Invalid I/O function code.
+303	-3	IE.DNR	Device not ready.

Table 14-5 (Cont.)
Return Status Summary

FORTRAN INTERFACE VALUE	ASSEMBLY LANGUAGE VALUE	ASSEMBLY LANGUAGE MNEMONIC	DEFINITION
+306	-6	IE.SPC	Illegal buffer.
+315	-15	IE.ABO	Request aborted.
+316	-16	IE.PRI	Privilege violation.
+317	-17	IE.RSU	Resource in use.
+319	-19	IE.BYT	Buffer address or length is odd.
+321	-21	IE.MOD	Illegal module number.
+322	-22	IE.CON	Another task already connected to interrupts.
+323	-23	IE.NOD	Insufficient dynamic memory to allocate secondary control block.
+379	-79	IE.NLK	Task not linked to ICS/ICR interrupts.
+380	-80	IE.FLN	ICR11 already offline.
+381	-81	IE.NST	Task is not installed.
+397	-97	IE.IEF	Invalid event flag number.

14.4.3 Optional Arguments

The calling sequences discussed in subsequent sections frequently contain optional arguments. These arguments are enclosed in square brackets within the calling sequence description. A statement containing such arguments may be written with these parameters deleted by truncating the argument list if the optional parameters are at the end of the calling sequence, or replacing them with commas if they are embedded elsewhere in the list. Consider the routine XYZ below having two optional arguments.

```
CALL XYZ(ibuf [,ilen] [,ival])
```

If the argument ival is to be omitted then the calling sequence would be:

```
CALL XYZ(IBUF,ILEN)
```

When an optional argument in the middle of the list is to be omitted it is replaced with a comma. Consider the routine XYZ, above. The following statement is used to omit the parameter ilen

```
CALL XYZ(IBUF,,IVAL)
```

14.4.4 Assigning Default Logical and Physical Units for ICS/ICR Input and Output - ASICLN/ASUDLN

The following subroutines must be called to assign and record a default LUN and physical unit if either parameter is to be unspecified in subsequent FORTRAN calls for which these parameters are optional.

Calling Sequence:

```
CALL ASICLN([lun] [,idsw] [,iunt])
CALL ASUDLN([lun] [,idsw] [,iunt])
```

Before a task can issue the call to ASUDLN, the ASN command must be issued through MCR to assign logical device UDnn to the appropriate physical ICS/ICR unit.

Argument Description:

lun - An optional integer variable whose value is the number of the LUN to be assigned to the physical unit specified by iunt or unit 0. If unspecified, no LUN is assigned.

idsw - An optional integer variable to receive the result of the assign lun directive.

iunt - An optional integer variable that specifies the unit number to be assigned. Assumed to be zero if omitted.

Return Status:

The following values are returned to idsw:

+1 - Assignment or function successfully completed.

-5 - LUN usage is interlocked because LUN is assigned to a device that is attached to another device or a file is currently open on the LUN

-96 - Invalid LUN

The call to ASUDLN assigns a LUN to logical device UD: and is provided for compatibility with existing UDC11 software. The call to ASICLN assigns a LUN to device IC:.

Upon successful issuance of the Assign LUN directive, the subroutine executes a Get LUN Information directive to obtain the actual unit numbers to be saved. It is therefore possible to alter the default physical unit referenced in a direct access call, by means of the ASN MCR function, provided that such logical assignments are done before the task is made active.

Examples:

1. Assign LUN 5 to ICR unit 3.

```
     CALL ASICLN (5,IERR,3)
     IF(IERR) 20,10,10
10   --------------
```

2. Assign LUN 1 to logical device UD:, unit 0

   a. The following MCR command is issued to create logical device UD0:, and assign all references to physical device IC1:.

      >ASN IC1: = UD:

   b. The FORTRAN call

      CALL ASUDLN (1)

      assigns logical device UD0: to LUN 1. Because of the previous ASN command the Executive will assign this LUN to physical device IC1: and return a value of one (1) for the unit number in response to the GET LUN Information directive. This value will be stored and later referenced whenever the physical unit number is unspecified in any of the FORTRAN calls that reference the I/O page directly.

14.4.5 Analog Input

The following routines provide the capability of performing A/D input:

AIRD/AIRDW - ISA Standard call to read multiple channels in random order. This call requires one or more control variables containing A/D channel and gain in the format shown in Table 14-6.

AISQ/AISQW - ISA Standard call to read multiple channels in sequential order.

Table 14-6
A/D Conversion Control Word

Bits	Meaning	IAD-IA
0-11	Channel number	Range: 0-1919
12-15	Gain value for this sample, expressed as a bit pattern as follows	Gain:

15	14	13	12	
0	0	0	0	1
0	0	0	1	2
0	0	1	0	illegal
0	0	1	1	illegal
0	1	0	0	10
0	1	0	1	20
0	1	1	0	illegal
0	1	1	1	illegal
1	0	0	0	50
1	0	0	1	100
1	0	1	0	illegal
1	0	1	1	illegal
1	1	0	0	200
1	1	0	1	1000
1	1	1	0	illegal
1	1	1	1	illegal

14.4.5.1 AIRD/AIRDW: Analog Input-Random Channel Sequence - The ISA standard call provides the capability of reading multiple A/D channels in random sequence.

```
CALL AIRD(inm,icont,idata[,isb],lun)
```

or

```
CALL AIRDW(inm,icont,...etc.)
```

Argument Descriptions:

inm - Integer variable specifying the number of channels to be read.

icont - An integer array of size inm containing control data in the format shown in table 14-6.

idat - An integer array of dimension inm to receive the converted values. Each element in the array is paired with a control element in icont that defines the channel and gain.

isb - An optional 2-word integer array to receive the results of the call as follows:

+1 - Conversion successfully completed. The second word contains the number of channels converted.

+3 - Number of channels requested was zero.

+4 - Insufficient dynamic storage to allocate I/O packet.

+8 - LUN was not assigned.

+99 - Invalid LUN.

+301 - At least one invalid control word was specified. The second I/O status word contains the number of channels successfully converted.

+303 - Device not ready. Interrupt response was not received from an A/D channel within one second after initiation. The second word of I/O status contains the number of channels successfully converted.

+306 - Control or data buffer not wholly within the user's addressing space.

+319 - Control or data buffer is byte aligned.

lun - An integer variable specifying the ICS/ICR logical unit number. This parameter is required.

Example:

The following example illustrates how A/D throughput can be enhanced when several IAD-IA A/D Converters are in a system. This is accomplished by means of interleaved samples that initiate parallel conversions on each module. Samples are to be obtained from 12 channels on three IAD-IA A/D converter modules at a gain of 1.

```
C
C PROGRAM TO SAMPLE 12 A/D CHANNELS
C IN RANDOM SEQUENCE FOR MAXIMUM
C THRUPUT.
C
C CHANNELS TO BE SAMPLED:
C
C         0
C         1      -A/D MODULE 0
C         2
C         3
C       120
C       121      -A/D MODULE 1
C       122
C       123
C       240
C       241      -A/D MODULE 2
C       242
C       243
C
C INTERLEAVED SEQUENCE FOR MAXIMUM
C THRUPUT.
C
C         0
C       120
C       240
```

```
C         1
C       121
C       241
C         2
C       122
C       242
C         3
C       123
C       243
C
C THE FORTRAN CONVENTION FOR ARRAY
C STORAGE CAN BE USED TO REPRESENT
C THE ABOVE SEQUENCE IN AN N X I INTEGER
C CONTROL ARRAY.  WHERE:
C
C       N = NUMBER OF MODULES TO BE SAMPLED
C       I = NUMBER OF SAMPLES PER/MODULE
C
C ALLOCATE STORAGE FOR CONTROL ARRAY
C
        DIMENSION ICONT (3,4)
C
C INITIALIZE CONTROL ARRAY FOR IAD-IA MODULE 0
C
        DATA ICONT(1,1),ICONT(1,2),ICONT(1,3),ICONT(1,4)/0,1,2,3/
C
C INITIALIZE CONTROL ARRAY FOR IAD-IA MODULE 1
C
        DATA ICONT(2,1),ICONT(2,2),ICONT(2,3),ICONT(2,4)/120,121,122,123/
C
C INITIALIZE CONTROL ARRAY FOR IAD-IA MODULE 2
C
        DATA ICONT(3,1),ICONT(3,2),ICONT(3,3),ICONT(3,4)/240,241,242,243/
C
C ALLOCATE STORAGE FOR DATA ARRAY
C IN SIMILAR FASHION TO FACILITATE
C CHANNEL REFERENCES
C
        DIMENSION IDATA (3,4)
C
C       BEGIN EXECUTABLE STATEMENTS
C
                .
                .
                .
C
C INITIATE A/D SYNCHRONOUS CONVERSION ON LUN 3
C
        CALL AIRDW(12,ICONT,IDATA,,3)
                .
                .
                .
```

14.4.5.2 AISQ/AISQW: Analog Input-Sequential Channel Sequence

The ISA standard call described below provides the capability of sampling multiple A/D channels in sequential order. Channels are sampled in increments of one, beginning with the channel specified in icont(1).

```
CALL AISQ(inm,icont,idata [,isb],lun)
```

or

```
CALL AISQW(inm,icont...etc.)
```

Argument Descriptions:

inm - Integer variable specifying the number of elements to be read.

icont - An integer array of size inm containing initial channel in the first element only, and gain in the format shown in Table 14-6 in the remaining elements.

idat - An integer array of size inm to receive the converted values. Each element is paired with the corresponding control element in icont that defines the gain parameter.

Channels are sampled sequentially starting with the first channel specified in element 1 of icont.

isb - An optional 2-word integer array to receive the results of the call as follows:

+1	-	Conversion successfully completed. The second word contains the number of channels converted.
+3	-	Number of channels requested was zero
+4	-	Insufficient dynamic storage to allocate I/O packet
+8	-	LUN was not assigned
+99	-	Invalid LUN
+301	-	At least one invalid control word was specified. The second I/O status word contains the number of channels successfully converted.
+303	-	Device not ready. Interrupt response was not received from an A/D channel within one second after initiation. The second word of I/O status contains the number of channels successfully converted.
+306	-	Control or data buffer not wholly within the user's addressing space
+319	-	Control or data buffer is byte aligned

lun - An integer vaiable containing the logical unit number. This parameter is required.

Example:

The following example illustrates the procedure for sequential sampling. Five channels are converted at gains of 1, 2, 20, 50, and 1000, starting at channel 3.

```
C
C ALLOCATE SPACE FOR STATUS ARRA
C
        DIMENSION ISB (2)
C
C ALLOCATE SPACE FOR CONTROL ARRAY
C AND ESTABLISH INITIAL VALUES
C
        DIMENSION ICONT(5)
        DATA ICONT(1),ICONT(2),ICONT(3)/O000003,O010000,O050000/
        DATA ICONT(4),ICONT(5)/O100000,O150000/
C
C ALLOCATE SPACE FOR DATA ARRAY
C
        DIMENSION IDAT (5)
                .
                .
                .
C
C INITIATE SEQUENTIAL, ASYNCHRONOUS CONVERSION
C VIA LUN 1
C
        CALL AISQ(5,ICONT,IDAT,ISB,1)
10      IF(ISB(1).NE.0) GO TO 20

        (continue processing)
                .
                .
                .
C
C TEST CONVERSION STATUS
C
        GO TO 10
20      (test for errors or process converted data)
                .
                .
                .
        END
```

14.4.6 Analog Output - AO/AOW: Multi-Channel

This ISA standard routine is called to output voltage from multiple D/A channels.

Calling Sequence:

```
        CALL AO(inm,icnt,idat[,isb][,lun])
```

or

```
        CALL AOW(inm,icnt...etc.)
```

Argument Descriptions:

inm - Integer variable containing the number of channels to be output.

icnt - Integer array containing the channel numbers to receive output.

idat - Integer array containing the output voltage setting as a value between 0 and 1023 where:

0 = 0 volts dc and

1023 = +9.99 volts (full scale).

isb - Optional 2-word integer array to receive status. One of the following values is returned in isb(1). The second element is always zero.

+1 - Function successfully completed.

+3 - Zero channels requested.

+4 - Insufficient dynamic storage to allocate an I/O packet.

+8 - LUN was not assigned.

+99 - Invalid LUN.

+303 - Controller not ready.

+321 - Nonexistent channel specified.

lun - Optional integer variable containing the logical unit number.

Example:

Output the variable voltages contained in IV(1) and IV(2) to D/A channels 2 and 3 respectively.

```
C
C ALLOCATE DATA ARRAY
C
       DIMENSION IV(2)
C
C ALLOCATE CONTROL ARRAY
C
       DIMENSION ICNT(2)
C
C ALLOCATE STATUS ARRAY
C
       DIMENSION ISB(2)
C
C INITIALIZE CONTROL ARRAY
C
       DATA ICNT(1),ICNT(2)/2,3/
                .
                .
                .
C
C PERFORM A/D OUTPUT VIA LUN 3
C
       CALL AOW(2,ICNT,IV,ISB,3)
       IF (ISB(1).GE.3) go to error processor
```

14.4.7 Digital Output - DOL/DOLW: Bistable Multiple Fields

The following ISA standard call provides the capability of latching or unlatching multiple 16-point bistable digital output fields.

Calling Sequence

```
CALL DOL(inm,icnt,idat,imsk[,isb][,lun])
```

or

```
CALL DOLW(inm,icnt...etc.)
```

Argument Descriptions:

inm - Integer variable specifying the number of fields to be latched or unlatched.

icnt - Integer array containing the initial point within each field.

idat - Integer array containing binary data that defines points within the field to be latched or unlatched. The state of each bit is interpreted as follows:

1 = latch point.

0 = unlatch point.

imsk - Integer array containing binary data that defines points within the field for which a change of state is permitted.

A bit set to 1 defines a point that may assume the state defined by the corresponding bit in idat. A 0 bit specifies a point for which no change of state is permitted.

isb - Optional 2-word integer array to receive the results of the call. Status is returned in isb(1) as shown below. isb(2) is always zero.

- +1 - Function successfully completed.
- +3 - Zero points specified.
- +4 - Insufficient dynamic storage to allocate an I/O packet.
- +8 - LUN not assigned.
- +99 - Invalid LUN.
- +303 - Controller not ready.
- +321 - Nonexistent point number specified. One or more points within the field do not exist.

lun - Optional integer specifying the Logical Unit Number.

Example:

Reset points 0,1,20 and 21

```
        DIMENSION ICNT(2),IDAT(2),IMSK(2)
C
C INITIALIZE THE CONTROL ARRAY
C
        DATA ICNT(1),ICNT(2)/0,20/
C
C INITIALIZE MASK ARRAY TO EFFECT A
C CHANGE-OF-STATE ONLY ON THE SPECIFIED
C POINTS.
C
        DATA IMSK(1),IMSK(2)/O00003,O000003/
                .
                .
                .
C
C RESET THE SPECIFIED POINTS.  ICR IS ASSIGNED
C TO LUN 3.
C
        CALL DOLW(2,ICNT,IDAT,IMSK,,3)
                .
                .
                .
```

14.4.8 Digital Input

Both of the following subroutines perform their functions through direct access to the ICS/ICR hardware registers. Therefore, the physical unit number replaces LUN in the calling sequences described below. Note that any need for conversion of BCD encoded digital input into binary, can be accomplished through the FORTRAN function

IBIN=KBCD2B (IBCD).

Binary data can be converted to BCD through the FORTRAN function

IBCD=KB2BCD (IBIN).

NOTE

When the physical unit number is explicitly included in the calling sequence, it cannot be reassigned by the MCR command ASN.

14.4.8.1 DI/DIW: Digital Input - Digital Sense Multiple Fields

- This ISA standard subroutine provides the capability of reading multiple 16-point contact sense fields.

Calling Sequence:

```
CALL DI(inm,icnt,idat[,isb][,iun])
```

or

```
CALL DIW(inm,icnt...etc.)
```

Argument Descriptions:

inm - Integer variable specifying the number of fields to be read.

icnt - Integer array containing the initial point number of each field.

idat - Integer array to receive the input data

isb - Optional, 2-word integer array to receive the results of the call. The status is returned in isb (1) as follows:

+1 - Function succesfully completed.

+3 - Zero points requested.

+321 - Nonexistent point requested. One or more points within the 16-bit field does not exist.

iun - Optional integer variable specifying the physical unit number.

Example:

Read two contact sense fields starting at points 3 and 27 on physical unit IC2:.

```
DIMENSION  ICNT(2), IDAT(2),ISB(2)
DATA   ICNT(1),ICNT(2)/3,27/
       .
       .
       .
CALL DI (2,ICNT,IDAT,ISB,2)
IF (ISB(1).GE.3) go to error procedure
       .
       .
       .
```

14.4.8.2 RCIPT: Digital Input - Digital Interrupt Single-Point - The following subroutine returns the state of a single digital interrupt point as a logical value.

Calling Sequence:

```
CALL RCIPT (ipt,isb[,iun])
```

Argument Descriptions:

ipt - Integer variable defining the point to be read.

isb - 2-word integer array to receive status and data as follows. Status is returned to isb(1).

+1 - Function successfully completed. Data is returned to isb(2) as a logical value, where:

.TRUE. (-1) = Point closed.

.FALSE. (0) = Point open.

+321 - Nonexistent point specified.

iun - Optional integer variable defining the physical unit number.

Example:

Read the state of contact interrupt point 3 on unit 0.

```
     DIMENSION ISB (2)
          .
          .
          .
     CALL RCIPT (3,ISB,0)
     IF (ISB(2).EQ..FALSE.) go to point open routine.
```

14.4.9 Digital Output Momentary - DOM/DOMW: Multiple Fields

This ISA standard call allows multiple 16-bit fields to be pulsed.

Calling Sequence:

```
     CALL DOM (inm,icnt,idat[,idx][,isb][,lun])
```
or
```
     CALL DOMW (inm,icnt...etc.)
```

Argument Descriptions:

inm - Integer variable specifying the number of fields to be pulsed.

icnt - Integer array containing the initial point in each field.

idat - Integer array defining the points to be pulsed. A bit is set corresponding to each point that is to be triggered.

idx - Optional dummy integer variable retained for compatibility with the standard form of the call.

isb - Optional 2-word integer array to receive the results of the call as follows in isb(1), isb(2) is set to zero.

+1 - Function successfully completed.

+3 - Number of fields to be output is zero.

+4 - Insufficient dynamic storage to allocate on I/O packet.

+8 - LUN not assigned.

+99 - Invalid LUN.

+303 - Controller not ready

+321 - Nonexistent point specified. One or more points within a field do not exist.

lun - Optional integer variable defining the logical unit number.

Example:

Pulse momentary digital output fields defined by points 20, 37 and 0 on LUN 1.

```
DIMENSION ICONT(3),IDAT(3)

DATA ICONT(1),ICONT(2),ICONT(3)/20,37,0/

CALL DOM(3,ICONT,IDAT,,1)
```

14.4.10 Remote Terminal Output - RTO/RTOW

The following function provides the capability of transmitting a character string to a remote ICR11 terminal. Both synchronous and asynchronous forms are supported.

```
CALL RTO (ibc,idat[,isb][,lun])
```

or

```
CALL RTOW (ibc,idat....etc.)
```

Argument Descriptions:

ibc - Integer variable specifying the number of bytes to output.

idat - Byte array (LOGICAL * 1) containing the character string to be output.

isb - Optional, 2-word integer array to receive the results of the call in isb(1) as follows. isb(2) is set to the number of bytes actually transferred to the device.

0 - Operation pending.

+1 - Function successfully completed.

+3 - Zero bytes to be transmitted.

+4 - Insufficient dynamic storage to allocate I/O packet.

+8 - Unassigned LUN.

+99 - Invalid LUN.

+303 - Device not ready. Terminal failed to respond with BUFFER EMPTY within 1 second after character was transmitted.

+306 - Part or all of buffer is out of the issuing task's addressing space.

+321 - Nonexistent module. Device is ICS11.

lun - Integer variable defining the logical unit number.

Example:

Output a character string to a remote terminal via the ICR unit assigned to LUN 3.

```
CALL RTOW(32,'APPLY +5 VOLTS TO A/D CHANNEL 10',,3)
```

14.4.11 Unsolicited Interrupt Data - Continual Monitoring

Subroutines are provided, that permit a FORTRAN program to continually monitor unsolicited interrupt data supplied to a user circular buffer as described in paragraph 14.3.6. Such routines allow the program to connect a buffer for input, disconnect the buffer upon completion and read and return the buffer contents in a format suitable for FORTRAN processing. The calls summarized below perform these functions for interrupting digital input modules, counters, and remote terminal inputs:

Interrupting Digital Inputs

CTDI - Connect a buffer to receive digital interrupts.

RDDI - Read the state of a single interrupting point.

RDCS - Read the state of a single interrupting point for which a change of state has been detected.

RDWD - Read 16 bits of interrupt data from the circular buffer.

DFDI - Disconnect a buffer from digital interrupts.

Counter Modules

CTTI - Connect a buffer to receive counter interrupts.

RDTI - Read the counter circular buffer.

DFTI - Disconnect a buffer from counter interrupts

Remote Terminal Input

CTTY - Connect a buffer to receive remote terminal inputs.

RDTY - Read remote terminal data from the circular buffer.

DFTY - Disconnect a buffer from remote terminal interrupts.

14.4.11.1 CTDI: Connect a Buffer for Receiving Digital Interrupt Data - The following routine allows a task to provide a circular buffer that will receive digital interrupt data, and define an event flag that will be set upon the occurrence of each interrupt.

Calling Sequence:

```
CALL CTDI (ibuf,isz,iev[,isb][,lun])
```

Argument Descriptions:

ibuf - An integer array making up the circular buffer that is to receive interrupt data.

isz - Integer variable specifying the length of the circular buffer in words.

iev - Integer variable specifying the event flag that is to be set whenever the driver receives an interrupt from a digital input module.

isb - Optional, 2-word integer array to receive the results of the call. The status values specified below are returned to isb(1).

+1 - Function successfully completed. isb(2) receives the number of words passed per interrupt in the low byte.

+4 - Insufficient dynamic storage to allocate an I/O packet.

+8 - Unassigned LUN.

+99 - Invalid LUN.

+306 - Part of buffer is out of the user's address space or buffer is too small to accommodate a single entry.

+316 - Privilege violation - task is checkpointable and not fixed in memory.

+319 - Buffer address or length is an odd number of bytes.

+322 - Another task is already connected to interrupts.

+397 - Invalid event flag specified

lun - Integer variable specifying the logical unit number.

The space allocated for the circular buffer must be large enough to accommodate at least one 5-word entry plus an additional 10 words of storage that are required by the subroutines that read circular buffer contents. Thus the buffer allocation specified by the integer variable isz may be computed as

isz = (10 + 5 * n)

where n is the number of entries to be contained in the buffer and isz is expressed in words.

14.4.11.2 Reading Digital Interrupt Data - Each of the following routines reads data that has been stored in the circular buffer and performs the following common processing:

1. Detects, and optionally reports, the occurrence of an error entry that has been placed in the buffer by the driver because of a nonrecoverable device fault (e.g., fatal serial line error or remote power-fail).

2. Clears the trigger event flag when no further entries remain to be processed.

3. Clears and optionally reports any overrun conditions.

Only one of the following three routines can be invoked by a single task.

a. RDDI: Read Digital Interrupt Data from a Circular Buffer

The RDDI FORTRAN subroutine reads contact interrupt data from a circular buffer that was specified in a CTDI call (see 14.4.11.1 above). It does no actual input or output, but rather performs a point-by-point scan of an interrupt entry in the buffer, returning the state of each point as a logical value.

On the initial call to RDDI, the module number and data of the next interrupt entry are read from the circular buffer and stored for subsequent reference. The subroutine then sets the current data bit number n to zero, examines the state of data bit n, and converts bit n to a point number via the following formula:

ipt = module number * 16 + n

On each subsequent call, n is incremented by one and then data-bit n is examined in the stored module data. When n reaches 16, it is reset to zero and an attempt is made to read the next interrupt entry from the circular buffer. If a valid entry is not found, ipt is set negative and ict (if specified) is either assigned a value of zero or an overrun count that is maintained by the ICS/ICR driver. If ict is

zero, no further entries remain. A nonzero value indicates that the driver received more data than could be stored in the buffer, and ict represents the number of entries that were discarded.

The variable ict, receives the control register contents that are set by the driver as described in section 14.3.6 whenever a nonrecoverable controller error occurs.

Calling Sequence:

```
CALL RDDI (ipt,ival[,ict])
```

Argument Descriptions:

ipt - is a variable to which the digital input point number is returned. It may be set as follows:

1. ipt < 0 if no valid entry is found

The specific value of ipt reflects the error that was detected as follows:

-1 - no data (i.e., no interrupt data currently in buffer)
-2 - overrun
-3 - hardware error

2. ipt => 0 if the value indicated is a point number; the state is returned to ival.

ival - is a variable to which the state of the point is returned; it may be set as follows:

1. .FALSE. (0) if the point is open

2. .TRUE. (-1) if the point is closed

ict - Optional integer variable to receive the overrun count. or the contents of the CSR register on the occurrence of a fatal controller error. Otherwise set to zero.

b. RDCS: Read Digital Interrupt Points That Have Changed State

The RDCS FORTRAN subroutine returns data in the format of subroutine RDDI as described above except that only points that have changed state are processed, resulting in significantly improved throughput and reduced processing overhead for the calling task.

Processing specific to the routine is as follows:

On the initial call, the module number, module data and change of state information are read from the circular buffer and stored for later reference. The subroutine then sets the current data bit number n to zero and begins scanning the change-of-state word until a nonzero bit is found. The point number and current state are then reported as previously described. If no change of state is found or when no further bits remain to be processed, the next entry is fetched as described above.

The processing of error conditions is identical to subroutine RDDI.

Calling Sequence:

```
CALL RDCS (ipt,ival[,ict])
```

Argument Descriptions:

ipt - Integer variable to receive the digital input point number. It may be set as follows:

1. ipt < 0 if no valid entry is found (i.e., overrun, error or no data in buffer). The specific value of ipt reflects the error that was detected as follows:

   -1 - no data
   -2 - overrun
   -3 - hardware error

2. ipt => 0 if the value indicated is a point number, the state is returned to ival.

ival - Integer variable to receive the state of the point as a logical value where:

1. .FALSE. (0) = point open

2. .TRUE. (-1) = point closed

ict - Optional integer variable. A nonzero value indicates that the variable has been set with an overrun count returned by the driver, or with the contents of the CSR register on the occurrence of a fatal controller error. Otherwise set to zero.

c. RDWD: Read a Full Word of Digital Interrupt Data

The following subroutine is called to return a full word of digital interrupt data from the circular buffer, and optionally change of state information. A new entry is read for each call; hence, throughput is high when processing is contingent upon several possible conditions within a module.

Calling Sequence:

```
CALL RDWD (imod,ival[,ict][,icos])
```

Argument Descriptions:

imod - an integer variable to receive the module number or status as follows:

1. imod < 0 if no data is present or an overrun condition or error was detected

   The specific value of ipt reflects the error that was detected as follows:

   -1 - no data
   -2 - overrun
   -3 - hardware error

2. imod => 0 Module number. Interrupt data is in ival

ival - Integer variable to receive the digital interrupt data.

ict - Optional integer variable. A nonzero value indicates that the variable has been set with an overrun count returned by the driver, or with the contents of the CSR register on the occurrence of a fatal error. Otherwise set to zero.

icos - Optional integer variable to receive change-of-state information. Bits set to a 1 correspond to points for which a change of state has been recorded.

14.4.11.3 DFDI: Disconnect a Buffer from Digital Interrupts - The following routine is called to disconnect a task's circular buffer from digital interrupts.

Calling Sequence:

```
CALL DFDI ([isb][,lun])
```

Argument Descriptions:

isb - Optional 2-word integer array to receive the results of the call as follows. isb(1) is always zero.

+1 - Function successfully completed.

+4 - Insufficient dynamic storage to allocate I/O packet.

+8 - Unassigned LUN.

+99 - Invalid LUN.

+322 - Task not connected to interrupts.

lun - Optional integer variable containing logical unit number.

14.4.11.4 CTTI: Connect a Buffer for Receiving Counter Data - The following subroutine may be called to connect a circular buffer that is to receive counter data and to define an event flag that is to be set upon occurrence of each interrupt.

Calling Sequence:

```
CALL CTTI (ibuf,isz,iev,iv[,isb][,lun])
```

Argument Descriptions:

ibuf - An integer array making up the circular buffer that is to receive interrupt data.

isz - Integer variable specifying the length of the circular buffer in words.

iev - Integer variable defining an event flag that is to be set whenever the driver receives an interrupt from a counter module.

iv - Integer array of initial counter values. One element is required for each counter in the physical unit. The value is used to initialize and reset the counter when a value of zero is reached. This parameter may be reset for a specific module through a call to SCTI.

isb - Optional 2-word integer array to receive the results of the call. The status values specified below are returned to isb(1).

+1 - Function successfully completed. isb(2) receives the number of words passed per interrupt in the low byte.

+4 - Insufficient dynamic storage to allocate an I/O packet.

+8 - Unassigned LUN.

+99 - Invalid LUN.

+303 - Controller not ready.

+306 - Part of buffer is out of the user's address space or buffer is too small to accommodate a single entry.

+316 - Privilege violation - task is checkpointable and not fixed in memory.

+319 - Buffer address or length is an odd number of bytes.

+322 - Another task is already connected to interrupts.

+397 - Invalid event flag specified.

lun - Integer variable specifying the logical unit number.

The space allocated for the circular buffer must be large enough to accommodate at least one 4-word entry plus an additional 8 words of storage required by the subroutine that reads buffer contents (RDTI). The buffer allocation specified by the variable isz may be computed as

```
isz = (8 + 4 * n)
```

where n is the number of entries to be contained in the buffer.

14.4.11.5 RDTI: Read Counter Data from the Circular Buffer - The following call returns counter interrupt data from the circular buffer. A new entry is read on each call.

Calling Sequence:

```
CALL RDTI (imod,ival[,ict])
```

Argument Descriptions:

imod - Integer variable to receive module number and status as follows:

1. imod < 0 No data in buffer, data overrun or error condition detected The specific value of ipt reflects the error that was detected as follows:

   -1 - no data
   -2 - overrun
   -3 - hardware error

2. imod => 0 Module number of counter. Interrupt data is in ival.

ival - Integer variable to receive the counter data at interrupt.

ict - Optional integer variable to receive the overrun count, or the ICSR contents returned by the driver on the occurrence of a fatal hardware error. Otherwise, set to zero.

14.4.11.6 Miscellaneous Counter Routines

a. RSTI: Read a Counter Module

The following routine directly accesses a counter register to return its current value.

Calling Sequence:

```
CALL RSTI (imod,isb,[,iun])
```

Argument Descriptions:

imod - An integer variable containing the number of the counter to be read.

isb - A two-word integer array to receive status and data as follows. Status is returned to isb(1).

+1 - Function successfully completed. Data is returned to isb(2).

+321 - Nonexistent module specified.

iun - Optional integer variable specifying the ICS/ICR physical unit number.

b. SCTI: Reset a Counter Initial Value

The following routine may be called by any task to revise the initial value that is used to activate a counter.

Calling Sequence:

```
CALL SCTI (imod,ival[,isb][,lun])
```

Argument Descriptions:

imod - Integer variable specifying the relative module number of the counter to be reset.

ival - Integer value specifying the new initial value.

isb - Optional 2-word integer array to receive status as follows. isb(2) is always zero.

+1 - Function successfully completed.

+4 - Insufficient dynamic storage to allocate an I/O packet.

+8 - Unassigned LUN.

+99 - Invalid LUN.

+303 - Controller not ready.

+321 - Nonexistent module specified.

lun - Optional integer specifying the logical unit number.

14.4.11.7 DFTI: Disconnect a Buffer from Counter Interrupts - The following subroutine is called to disconnect the task's circular buffer from interrupts.

Calling Sequence:

```
CALL DFTI ([isb][,lun])
```

Argument Descriptions:

isb - Optional 2-word integer array to receive status as follows. isb(2) is always zero.

+1 - Function successfully completed.

+4 - Insufficient dynamic storage to allocate an I/O packet.

+8 - Unassigned LUN.

+99 - Invalid LUN.

+322 - Task was not connected to interrupts.

lun - Optional integer variable specifying the Logical Unit Number.

14.4.11.8 CTTY: Connect a Circular Buffer to Terminal Interrupts - The following routine allows a task to provide a circular buffer to receive remote terminal input data, and to define an event flag that is set on the occurrence of each interrupt.

Calling Sequence:

```
CALL CTTY (ibuf,isz,iev[,isb][,lun])
```

Argument Descriptions:

The following arguments are identical in form and function to those described for subroutine CTDI (section 14.4.11.1).

ibuf - An integer array making up the circular buffer, that receives interrupt data.

isz - Length of the circular buffer in words.

iev - Event flag to be set on each terminal interrupt.

Buffer size is computed as

```
isz = (8 + 4 * n)
```

where n is the number of entries that can be stored in the buffer.

isb - Optional 2-word integer array to receive the results of the call. The status values specified below are returned to isb(1).

+1 - Function successfully completed. isb(2) receives the number of words passed per interrupt in the low byte.

+4 - Insufficient dynamic storage to allocate an I/O packet.

+8 - Unassigned LUN.

+99 - Invalid LUN.

+306 - Part of buffer is out of the user's address space or buffer is too small to accommodate a single entry.

+316 - Privilege violation - task is checkpointable and not fixed in memory.

+319 - Buffer address not on a word boundary or length is an odd number of bytes.

+321 - Nonexistent module specified. Unit is ICS11.

+322 - Another task is already connected to interrupt.

+397 - Invalid event flag specified.

lun - Logical unit number.

14.4.11.9 RDTY: Read a Character from the Terminal Buffer - This subroutine retrieves a single character from the terminal circular buffer on each call.

Calling Sequence:

```
CALL RDTY (ind,ichr[,ivr])
```

Argument Descriptions:

ind - An integer variable to receive status as follows:

1. =0 character retrieved from buffer is in ichr
2. <0 no data in buffer, overrun, or hardware error

The specific value of ind reflects the error that was detected as follows:

-1 - no data
-2 - overrun
-3 - hardware error

ichr - Logical * 1 or integer variable to receive the terminal data. If an integer is specified only the low byte will be set.

ivr - Optional integer variable to receive the overrun count, or the ICSR contents on the occurrence of a fatal hardware error. Otherwise set to zero.

14.4.11.10 DFTY: Disconnect a Circular Buffer from Terminal Input - The following routine disconnects a task's circular buffer from terminal inputs.

Calling Sequence:

```
CALL DFTY ([isb][,lun])
```

Argument Descriptions:

isb - Optional, 2-word integer array to receive status in isb(1) as follows. isb(2) is always set to zero.

+1 - Function successfully completed.

+4 - Insufficient dynamic storage to allocate an I/O packet.

+8 - Unassigned LUN.

+99 - Invalid LUN.

+322 - Task was not connected to interrupts.

lun - An optional integer array specifying the logical unit number.

14.4.11.11 Programming Example - The following are excerpts from a FORTRAN program that is to monitor a remote terminal for input and echo the received characters when a carriage return is detected.

```
C
C     SPECIFY BYTE FORMAT FOR TERMINAL DATA
C
      LOGICAL*1 TCHR
C
C     ALLOCATE STORAGE FOR THE TERMINAL
C     BUFFER
C
      DIMENSION IBUF(32)
C
C     ALLOCATE STORAGE FOR THE PACKED
C     INPUT DATA SO THAT IT IS ALIGNED
C     ON A WORD BOUNDARY
C
      DIMENSION ICHR(40)
      DIMENSION TCHR(80)
      EQUIVALENCE (TCHR,ICHR)
C
C     ALLOCATE STORAGE FOR A
C     2-WORD STATUS BLOCK
C
      DIMENSION ISB(2)
C
C     INITIALIZE ICR11 LOGICAL UNIT(7) AND
C     TRIGGER EVENT FLAG NUMBER(2)
C
      DATA IEV, LUN/2, 7/
           .
           .
           .
C
C     CONNECT THE TASK TO TERMINAL
C     INPUTS. IF CONNECT FAILS--STOP 1
C
      CALL CTTY (IBUF,32,IEV,ISB,LUN)
      IF (ISB(1).GE.3) STOP 1
C
C     10--POLL THE CIRCULAR BUFFER
C         FOR DATA. ECHO THE LINE WHEN
C         80 CHARACTERS ARE RECEIVED
C         OR A CARRIAGE RETURN IS
C         DETECTED.
C

10    DO 70 I = 1,80
C
C     20--WAIT FOR TRIGGER EVENT FLAG
C
20    CALL WAITFR (IEV)
C
C     30--PACK THE CIRCULAR BUFFER DATA
C         INTO THE BYTE ARRAY
C
30    CALL RDTY (ISB,TCHR(I), IVR)
C
C     DISPATCH ON ERROR CONDITION
C
      GO TO (20,50,40)-ISB
      GO TO 60
C
C     40--REPORT HARDWARE FAULT
```

```
C
40    CALL ALARM (IVR)
          .
          .
          .
      GO TO 30
C
C     50--REPORT OVERRUN CONDITION
C
50    CALL LOST (IVR)
          .
          .
          .
      GO TO 30
C
C     60--CHECK FOR CARRIAGE RETURN,
C         EXIT TO ECHO ROUTINE IF
C         PRESENT
C
60    IF (TCHR(I).EQ."15) GO TO 80

70    CONTINUE
C
C     80--FALL THROUGH TO ECHO A LINE
C
      CALL RTOW (I,TCHR,,LUN)
C
C     DISCONNECT TERMINAL BUFFER, EXIT
C
      CALL DFTY (,LUN)
      CALL EXIT
      END
```

The procedure for reading the buffer in the example above may be summarized as follows:

1. Wait for the trigger event flag specified in the call to connect the buffer.

2. Upon regaining control, call the appropriate routine to read the buffer until one of the following terminal conditions is detected:

   a. All data has been read,

   b. An overrun count is detected,

   c. A fatal error is encountered.

3. On the occurrence of 2a or 2b, perform any appropriate processing; then return to scan for additional data.

4. If a hardware error is detected, use the ICSR register contents for further fault analysis and warning as appropriate. In the event of such an error, the event flag will not be set by the driver again unless normal service is resumed.

5. The calling task should not execute the Wait-For directive until the buffer-empty condition is detected.

14.4.12 Unsolicited Interrupt Processing - Task Activation

The following routines provide the capability of linking a task to an interrupt, soliciting information from the driver concerning how the task was activated, and unlinking a task from all interrupts.

14.4.12.1 LNK: Link a Task to Interrupts - This subroutine allows any installed task to be activated on the occurrence of any unsolicited interrupt.

Calling Sequence:

```
CALL LNK (tnam,iprm[,isb][,lun])
```

Argument Descriptions:

tnam - Real variable containing task name in RADIX-50 format.

iprm - 5-word integer array containing the following data:

iprm(1) - Interrupt class. May be one of the following:

0 - Digital interrupts

1 - Counters

2 - Remote terminal (Control-C only)

3 - Error interrupts.

iprm(2) - Reserved

iprm(3) - Optional event flag set if task to be activated is not dormant when the interrupt occurs.

iprm(4) - Hardware-dependent parameters as follows:
iprm(5)

Interrupt Class	Parameter Contents
Digital	iprm(4) = Point number iprm(5) = Change-of-state mask
Counter	iprm(4) = Module number iprm(5) = Counter initial value
Remote Terminal	iprm(4) = not used
Error	iprm(5) = not used

isb - Optional 2-word integer array to receive status in isb(1) as follows. isb(2) is always set to zero.

+1 - Function successfully completed.

+3 - Unrecognized interrupt class specified.

+4 - Insufficient dynamic storage to allocate I/O packet.

+8 - Unassigned LUN.

+99 - Invalid LUN.

+301 - Task tnam not installed.

+303 - Controller not ready.

+317 - Resource in use. Other task already linked to interrupt.

+323 - Insufficient dynamic memory to allocate secondary control block.

+397 - Invalid event flag number specified.

lun - Optional integer specifying the logical unit number.

Example:

Link task ALARM to report fatal hardware errors arising from a malfunction on any ICR11 physical unit.

```
      DIMENSION IPRM(5)
      C
      C  INITIALIZE PARAMETER ARRAY WITH:
      C    1.  OPERATION CODE
      C    2.  RESERVED ELEMENT CLEARED
      C    3.  GLOBAL EVENT FLAG
      C

      DATA IPRM(1), IPRM(2), IPRM(3)/3,0,64/

      DATA ALARM/6RALARM /
           .
           .
           .
      CALL LNK (ALARM,IPRM,,7)
           .
           .
           .
```

14.4.12.2 RDACT: Read Activation Data - The following call allows a task to determine the interrupt conditions that caused it to become active.

Calling Sequence:

```
CALL RDACT (iprm[,isb][,lun])
```

Argument Descriptions:

iprm - 6-word integer array to receive activatin data in the following format.

iprm(1) - Activation indicator (see section 14.3.7.5).

iprm(2) - Physical unit number of ICR.

iprm(3) - Generic code. Set to one of the following values.

0 - Remote terminal

1,2,3 - Digital interrupt

4,5,6 - Counter interrupt

177770 - Fatal hardware error

iprm(4) - Relative module number.

iprm(5) - Hardware-dependent data.
iprm(6)

The following data is returned based upon the type of interrupt module.

Module Type	Generic Code	Parameter Contents
Remote Terminal	0	iprm(5) = terminal input character iprm(6) = undefined
Digital Interrupt	1,2,3	iprm(5) = module data iprm(6) = change-of-state data
Counter	4,5,6	iprm(5) = value of the counter at interrupt iprm(6) = undefined
Error	177770	iprm(5) = contents of ICSR iprm(6) = contents of ICAR.

isb - Optional 2-word integer array to receive status in isb(1) as follows. isb(2) is set to zero.

+1 - Function successfully completed.

+4 - Insufficient dynamic storage to allocate I/O packet.

+8 - Unassigned LUN.

+99 - Invalid LUN.

+306 - iprm array not fully within the task's addressing space.

+319 - Address of iprm is odd.

+379 - Task not linked to ICS/ICR interrupts.

lun - Optional integer variable specifying the logical unit number.

Example:

The following is an excerpt from a program that reads activating data into array IACT and conditionally exits if the event flag (IEFN) specified in a previous link request is not set.

```
C
C     ALLOCATE SPACE FOR DATA ARRAY
C
      DIMENSION IACT(6)
            .
            .
            .
10    CALL RDACT (IACT,,7)
            .
            .
            .
C
C     CLOSE ALL FILES
C
      CALL CLOSE(1)
      CALL CLOSE(2)
C
C     EXIT IF TRIGGER EVENT FLAG IS NOT SET
C     ELSE CLEAR EVENT FLAG AND RESTART.
C
      CALL EXITIF (IEFN)
C
C     FLAG WAS SET.  CLEAR IT AND
C     CONTINUE.
C
      CALL CLREF (IEFN)
      GO TO 10
      STOP
      END
```

The foregoing example illustrates the following considerations when a task is made active by ICS/ICR interrupts:

1. To avoid race conditions, the Exit-If directive should be used to test the state of the event flag and conditionally exit. Issuing a Test Event Flag directive followed by an Exit would cause a flag set condition occurring after the test to go unrecognized.

2. Use of the Exit-If directive bypasses the closure of all files that is normally done automatically by the FORTRAN object time system when the program executes a STOP or CALL EXIT statement. Thus, to exit cleanly, the program must explicitly close all files before invoking the directive.

14.4.12.3 UNLNK: Remove Interrupt Linkage to a Task - The following call removes all linkage between a task and ICS/ICR interrupts.

Calling Sequence:

```
CALL UNLNK (tnam,iprm,[isb],[lun])
```

Argument Descriptions:

tnam - Real variable containing task name in Radix-50 format.

iprm - Integer variable containing the interrupt class. May be one of the following:

0 - Digital interrupts

1 - Counters

2 - Remote terminal

3 - Error interrupts

4 - All interrupts.

isb - Optional, 2-word integer array to receive the results of the call in isb(1) as follows. isb(2) is set to zero.

+1 - Function successfully completed.

+4 - Insufficient dynamic storage to allocate an I/O packet.

+8 - Unassigned LUN.

+99 - Invalid LUN.

+379 - Task not linked to ICS/ICR interrupts.

+380 - Task not installed.

lun - Integer variable specifying the logical unit number.

Example:

Remove the linkage between task ALARM and all ICS/ICR interrupts.

```
DATA ALARM/6RALARM /

CALL UNLNK (ALARM,,,7)
```

14.4.13 Maintenance Functions

The following functions cause the ICS/ICR driver to suppress or enable hardware error reporting while online maintenance and troubleshooting is in progress as described in paragraph 14.3.9.

OFLIN - Place selected unit offline.

ONLIN - Return selected unit to online status.

These calls may be issued only by a privileged task.

14.4.13.1 OFLIN: Place Selected Unit in Offline Status - The following call is executed to set a controller offline:

```
CALL OFLIN ([isb] [,lun])
```

Argument Descriptions:

isb - Optional 2-word integer array to receive the results of the call in isb(1) as follows. isb(2) is always zero.

+1 - Function successfully completed.

+4 - Insufficient dynamic storage to allocate an I/O packet.

+8 - LUN not assigned.

+99 - Invalid LUN.

+316 - Issuing task not privileged.

+380 - Device already offline.

lun - Optional integer variable specifying the ICS/ICR logical unit number.

14.4.13.2 ONLIN: Return a Device to Online Status - The following call will return the selected unit to online status.

```
CALL ONLIN ([isb] [,lun])
```

Argument Descriptions:

isb - Optional 2-word integer array to receive the results of the call in isb(1) as follows. isb(2) is always zero.

+1 - Function successfully completed.

+4 - Insufficient dynamic storage to allocate an I/O packet.

+8 - LUN not assigned.

+99 - Invalid LUN.

+316 - Issuing task not privileged.

14.5 ERROR DETECTION AND RECOVERY

Error Detection and recovery procedures encompass the following contingencies.

1. Nonrecoverable serial line errors
2. Power-fail at the remote station
3. Power recovery at the processor
4. No response from an interrupting module

The first two conditions are dealt with in a manner similar to other types of unsolicited interrupts. Specifically, such occurrences may cause a task to be activated, and are reported to all tasks that are connected to digital, counter or terminal input. The following paragraphs discuss specific driver activity relating to each error condition.

14.5.1 Serial Line Errors

The driver detects nonrecoverable serial line errors. A nonrecoverable error condition is defined as the occurrence of a predetermined number of error interrupts in an interval of 1 second or no response from the controller upon initiation of an output data transfer via the serial line. The occurrence of such a condition causes the driver to perform as follows:

1. Place the controller in a "not ready" status
2. Disable further error interrupts
3. Report the condition to the task that is linked to errors, and to any tasks connected to receive unsolicited interrupt data from the faulty unit. Subsequent QIO requests that transfer data to or from the unit are rejected with a status of IE.DNR.

Requests for interrupting modules that are pending (A/D converters and terminal output) are allowed to time out with the error code IE.DNR. The serial line error rate required to consider the link inoperative may be specified by the user at the time of system generation.

After reporting the error as described above, the driver will automatically remove the "not ready" status when the error condition is not detected at the end of any 1-second interval. If requested during system generation, the state of the following remote modules will be restored as described.

1. Bistable outputs - set to last recorded state
2. Counters - reinitialized to last initial value
3. Analog outputs - restored to last output value.

14.5.2 Power-fail at a Remote Site

The detection of AC low from the remote site will immediately trigger the processing described in section 14.5.1. The absence of AC-low will automatically return the unit to the "ready" status.

If specified, the state of the following remote modules will then be restored as described:

1. Bistable outputs - set to last recorded state
2. Counters - reinitialized to last initial value
3. Analog outputs - restored to last output value.

14.5.3 Power Recovery at the Processor

Power recovery by the processor will initiate the activity described in section 14.5.2 for both local and remote file boxes. However, power recovery processing at the processor will not be reported to a task that is linked to error interrupts or connected to receive unsolicited interrupt data.

14.5.4 Unit in Offline Status

A unit that is offline (see section 14.3.9.1) is considered to be under manual control for purposes of diagnosis and maintenance. Under these conditions, error reporting as described in section 14.5.1 is unnecessary and frequently undesirable since fault indications are generally a by-product of these activities (i.e., a remote unit is shut down to install an I/O module) not the result of a genuine controller fault.

Furthermore, to permit the operation of diagnostic software, it is advisable to attempt to service all QIO requests regardless of the controller status. Consequently, under these circumstances, error reporting and detection are modified as follows when the controller is offline:

1. Access to the controller with the intention of transmitting data to or from the device is restricted to privileged tasks.

2. The task linked to error interrupts and any tasks receiving interrupt data are not notified of remote power-fail or fatal serial line errors.

3. All device error interrupts become disabled.

4. An attempt is made to service all QIO requests if issued by a privileged task. If such requests time out (i.e., A/D converter or remote terminal output), they are terminated with the error code IE.ABO rather than with IE.DNR. No hardware errors are reported for I/O requests that are normally completed immediately (e.g., bistable digital output).

14.5.5 Error Data - ICSR and ICAR Registers

Whenever a reportable error occurs, the driver returns the contents of the appropriate control and status register (ICSR) and, in some cases, the contents of the address register (ICAR) to assist in fault diagnosis. Tables 14-7 and 14-8 describe the contents of these registers.

Table 14-7
ICSR Contents

BIT	NAME	READ/WRITE	DESCRIPTION
15	OUTPUT BUSY	R	Indicates output buffer cannot accept new data.
14	MAINT	R/W	Maintenance.
13	NOT USED	R	Always set to 1.
12	ERROR	R	Indicates occurrence of communication serial line error. Reset when ICAR is read.

Table 14-7 (Cont.)
ICSR Contents

BIT	NAME	READ/WRITE	DESCRIPTION
11	MAINT	R/W	Maintenance.
10	PWR FAIL	R	Remote Power Supply AC LO indicator.
9	TBMT INT EN	R/W	Enables bit 15 of ICAR to interrupt.
8	MAINT	R/W	Maintenance.
7	MOD INT	R	Indicates I/O Module requires interrupt servicing.
6	RESET	W	Resets all I/O modules. Always read as 0.
5	TTY ENABLE	R/W	Activates TTY mode, disables I/O mode.
4	PWR FAIL INT ENABLE	R/W	Enables bit 10 to interrupt.
3	BMT INT ENABLE	R/W	Enables complement of bit 15 to interrupt.
2	MOD INT ENABLE	R/W	Enables Bit 7 to interrupt.
1	ERROR INT ENABLE	R/W	Enables Bit 12 to interrupt.
0	RIF	R/W	Resets the interrupting module's flag when set and the module is addressed. This clearing action also resets the RIF bit.

Table 14-8
ICAR Contents

BIT	NAME	DESCRIPTION
15	TBMT	Indicates TTY output buffer can accept new data.
14	PCL	Pulse closed. This bit is set by a jumper on a digital interrupt module. This jumper is removed if contact closures are not of interest to the user.
13	POP	Pulse Opened. This bit is set by a jumper on a digital interrupt module. This jumper is removed if contacts opening are not of interest to the user.
12	DA	Indicates terminal character has been received. Cleared by reading terminal character.
11-08	Generic Code	A 4-Bit binary code that identifies the the type of module requesting the interrupt.
07-00	Module Address	8-Bit address of the module requesting the interrupt.

14.6 DIRECT ACCESS

Section 14.1.3 notes those ICS/ICR-11 functions that may be performed by referencing a module through its physical address in the I/O page. Under RSX-11M such access is accomplished by one of the following methods:

1. A privileged task or any task running in an unmapped system has unrestricted access to the I/O page and may therefore access each module by absolute address.

2. Using the Task Builder, a task may link to a global common area whose physical address limits span a set of locations in the I/O page. This method applies to either a mapped or unmapped system.

The latter method allows a task to be transported to any other system simply by relinking. Moreover, in a mapped system the memory management hardware aborts all references to device registers outside the physical address limits of the common block.

Because the software allows arbitrary module placement, direct reference, in either case, must be accomplished by translating a relative module number to a physical or virtual register address within the I/O page. This translation or mapping is performed by means of a table (ICTAB.MAP) that is created during system generation, and inserted in the system object module library.

The operations required to implement each method may be summarized as follows:

1. Unrestricted access to the I/O page

   a. Based upon the user's response to the ICS/ICR SYSGEN queries, the MACRO source file ICTAB.MAC is automatically created nder UIC [11,10] on the source disk. This file contains tables that describe the physical location of each counter, digital interrupt, and digital sense module in the target system.

   b. ICTAB.MAC is assembled for eventual inclusion in the system object module library.

   c. The MACRO source file ICOM.MAC, under UIC [11,10] on the source disk, is assembled to generate global definitions for the first ICS/ICR address on the I/O page and the number of ICS/ICR controllers in the target system. The resulting object file is incorporated in the system library file.

   d. A task is built containing the appropriate global references. Such references are resolved when the Task Builder automatically searches the system library.

Steps a, b, and c are executed once. Step d is performed each time a task that references the ICS/ICR-11 is created.

2. Access to the I/O page through a Global Common Block:

   a. Steps 1a and 1b are performed.

   b. File ICOM.MAC under UIC [11,10] is assembled to define

the first ICS/ICR module address as a relocatable value, the number of I/O page locations required, and the number of controllers present on the target system.

c. File ICOM.OBJ, created in step b, is linked using the Task Builder to create an image of the Device Common Block on Disk.

d. The SET and INSTALL MCR or VMR commands are used to allocate space for the common block and declare the block resident in the target system.

e. A task is created containing the appropriate global references to the common block and mapping table. Common block references are resolved by directing the Task Builder to link the Task to the device common block (ICOM). The mapping table reference is resolved from the system library module ICTAB.

The detailed procedure for creating the necessary object files and device common block is performed automatically as part of the system generation process, and is described fully in the RSX-11M System Generation Manual (DEC-11-OMGIA-C-D) (SYSGEN). Therefore, the discussion in the following paragraphs is limited to procedures for linking to the device common block, and using the file ICTAB.MAC to determine module addresses within the I/O page.

NOTE

ICS/ICR inputs are not valid until 3ms after power recovery at the processor. Tasks that are referencing inputs directly may establish a power recovery AST entry point that suspends task execution for the necessary time interval.

14.6.1 Linking a Task to the ICS/ICR Common Block

Once the device common block has been created, a task may access ICS/ICR modules by linking to the common block. This can be done by using the Task Builder commands shown in the following example.

```
TKB>TASK,LP:=TASK.OBJ
TKB>/
ENTER OPTIONS:
TKB> COMMON=ICOM:RO
TKB>/
```

The illustration is valid for either a mapped or unmapped system. In both cases the Task Builder links the task to the common block by relocating the global symbol definitions contained in the common block symbol table file ICOM.STB located under UIC [1,1]. If memory management is present, the Executive will map the appropriate physical locations into the task's virtual addressing space when the task is made active.

14.6.2 Accessing the I/O Page

After the task has been linked to the I/O page, either directly or through reference to the device common block, access to specific ICS/ICR counter, or digital input modules during task execution is a three-step process:

1. The task generates a request for module data by specifying module type, relative module number and physical unit number.

2. The data contained in module ICTAB is accessed to translate the arguments of step 1 to a physical offset from the ICS/ICR base address on the I/O page.

3. The ICS/ICR base address, defined in the common block or system library module that was created from file ICOM.MAC, is added to the offset to compute a physical or virtual address and the module data is read.

The next few paragraphs describe the format of the system library module ICTAB, and common block module ICOM in detail. A sample MACRO subroutine that references these modules is then presented.

14.6.2.1 Mapping Table Format - The mapping table created by SYSGEN (file ICTAB.MAC) is used to translate module type, relative module number, and physical unit number for counter, digital interrupt, and digital sense modules, to the physical or virtual address of the module on the I/O page. This module must be assembled and inserted in the system object module library before the standard FORTRAN callable routines can be used to read digital input and counter modules. The table contains one set of entries for each physical unit. The entry sets are arranged in order of ascending unit number (Figure 14-1A). Entries within each unit are arranged in sequence by module type as shown in this figure.

The structure of each entry is depicted in Figure 14-1B. Entries are 18 bytes long. Byte 0 contains the highest number of modules of a given type that can be referenced for the controller. Bytes 2 through 17, when indexed by relative module numbers, yield a value between 0 and 255 representing the physical location of the module within the set of external page addresses allocated to the ICS/ICR-11.

The following global symbols are defined by this module:

.ICTAB = Location of mapping tables

I.CTBL = Length in bytes of one set of entries

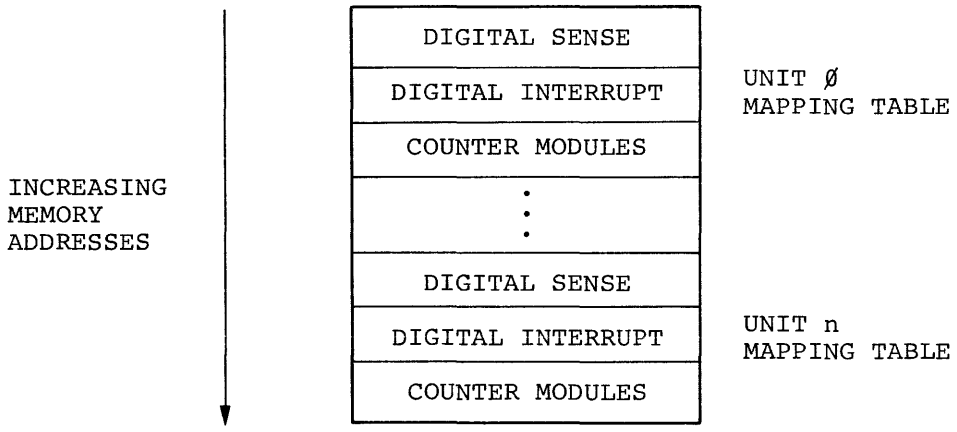


Figure 14-1A
Mapping Table Format

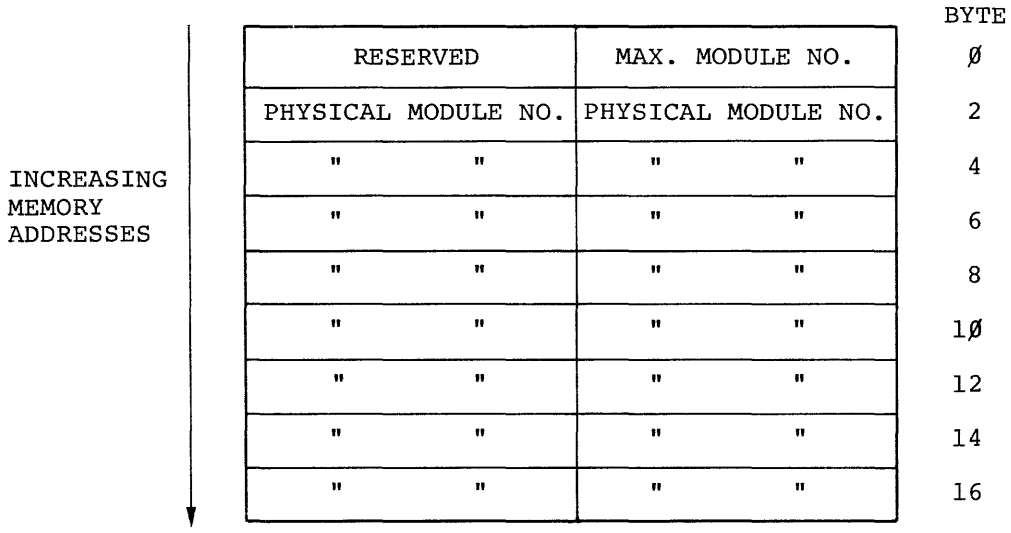


Figure 14-1B
Mapping Table Entry Format

14.6.2.2 I/O Page Global Definitions - As previously mentioned, module ICOM contains symbolic definitions for I/O page references that are resolved either through unrestricted access or by means of a device common block that is resident on the I/O page. The procedures for implementing either method are carried out during system generation. Upon completion, the following global symbols are defined and later referenced by the FORTRAN callable subroutines:

.ICMD = First ICS/ICR virtual or physical address within the I/O page.

I$$C11 = Number of ICS/ICR controllers

If the global common block was built, the definitions above are contained in the symbol table file that was created by the Task Builder; otherwise, they are included in the system object module library.*

14.6.2.3 Sample Subroutine - The following subroutine, residing in the system library, utilizes the modules previously described, to read ICS/ICR module data.

```
;
; READ ICS/ICR-11 DIRECT ACCESS INPUTS
;
; LOCAL DATA
;
; ADDRESS OF ICS/ICR-11 MAPPING TABLES
;
        .ENABL  LSB
N=0                                 ;
ICMAP:                              ;

        .REPT   12.

        .WORD   .ICTAB+<I.CTBL+N>
N=N+1

        .ENDR

;+
;
; **-.RDIC-READ ICS/ICR-11 DIRECT ACCESS INPUTS
;
; THIS SUBROUTINE IS CALLED TO TRANSLATE RELATIVE MODULE NUMBER
; TO PHYSICAL EXTERNAL PAGE ADDRESS AND READ THE MODULE DATA.
;
; INPUTS:
;
;       R0 = RELATIVE MODULE NUMBER
;       R1 = MODULE CODE
;
;               WHERE:
;                       0 = CONTACT SENSE
```

* The definitions are included in module ICOM in the system library or in the STB file ICOM.STB under UIC[1,1] on the system disk. The STB file is automatically referenced by the Task Builder in response to the use of the LIBR keyword.

```
;                       1 = CONTACT INTERRUPTS
;                       2 = COUNTERS
;
;       STACK SETUP IS AS FOLLOWS:
;               (SP)+00 = RETURN TO CALLER
;               (SP)+02 = I/O STATUS BLOCK ADDRESS (NOT REFERENCED).
;               (SP)+04 = PHYSICAL UNIT NUMBER
; OUTPUTS:
;
;       C/CLEAR
;
;               R0 = MODULE DATA
;
;       C/SET:
;
;               NONEXISTENT PHYSICAL UNIT NUMBER OR MODULE SPECIFIED
;
;-

.RDIC::                                 ;
        MOV     4(SP),R2                ; GET PHYSICAL UNIT NUMBER
        CMP     #I$$C11-1,R2            ; LEGAL UNIT NUMBER?
        BLO     10$                     ; IF LO NO
        ASL     R2                      ; CONVERT PHYSICAL UNIT NUMBER TO WORD
                                        ; OFFSET
        MOV     ICMAP(R2),R2            ; GET ADDRESS OF MAPPING TABLE
                                        ; ENTRIES FOR THIS UNIT
        ASL     R1                      ; CONVERT CODE TO WORD OFFSET
        ADD     R1,R2                   ; MULTIPLY OFFSET BY 9 AND ADD
                                        ; TO TABLE ADDRESS
        ASL     R1                      ; ...
        ASL     R1                      ; ...
        ASL     R1                      ; ...
        ADD     R1,R2                   ; COMPUTE OFFSET TO TABLE
        TSTB    (R2)                    ; MODULE EXIST?
        SEC                             ; ASSUME NO
        BEQ     10$                     ; IF EQ NO
        INCB    R0                      ; CONVERT TO NUMBER OF MODULES
        CMPB    (R2)+,R0                ; LEGAL MODULE NUMBER?
        BLO     10$                     ; IF LO NO
        INC     R2                      ; POINT TO TABLE ENTRIES
        ADD     R0,R2                   ; OFFSET TO MODULE NUMBER
        CLR     R0                      ; SET FOR MOVB WITHOUT SIGN EXTEND
        BISB    (R2),R0                 ; GET INDEX TO MODULE
        ASL     R0                      ; CONVERT TO WORD OFFSET
        MOV     .ICMD(R0),R0            ; GET MODULE DATA
10$:                                    ;
        RETURN                          ;

        .END
```

14.7 CONVERSION OF EXISTING SOFTWARE

The following paragraphs are intended as guidance in converting existing UDC or ICS software to run under the ICS/ICR-11 driver and associated FORTRAN support routines. The differences described here are restricted to module support and features that would affect existing software. New features, unsupported in previous systems, are not discussed.

14.7.1 Features

Principal features affecting existing software are:

1. Support for the ICS/ICR-11 as a multi-unit, multi-controller device.

2. Removal of software restrictions on the placement of functionally similar modules.

Multi-unit support affects any software that addresses modules outside the range of a single file box. In general, such software must be modified at the source level.

Unrestricted module placement affects MACRO-11 programs that directly access digital input and counter modules. Such programs may utilize the library routine described in section 14.3 to read data from these modules.

14.7.2 Module Support

14.7.2.1 IAD-IA A/D Converter and IMX-IA Multiplexer

MACRO Interface: Identical to UDC11 driver
FORTRAN Interface: Same as UDC11

Functional Differences:

The ICS/ICR-11 driver can initiate parallel conversions on each IAD-IA in a file box that is referenced by a single QIO request. The UDC11 driver performs all conversions serially.

The ICS/ICR-11 driver supports any permissible configuration of IAD-IA A/D converters and IMX-IA multiplexers. The UDC11 driver requires that eight module slots be reserved for each IAD-IA in the system regardless of the actual number of multiplexers installed.

14.7.2.2 16-Bit Binary Counter

MACRO Interface: Identical to UDC11 driver

FORTRAN Interface: Same as UDC11; however, if the counter is read through a call to RDTI then the task must be relinked to incorporate the revised FORTRAN Interface routine.

Functional differences:

The ICS/ICR-11 driver permits any task to reset an initial counter value (via FORTRAN call RSTI or through the IO.RTI QIO function). The UDC11 driver restricts this operation to a task that has connected to counter interrupts.

14.7.2.3 Bistable Digital Output

MACRO Interface: Identical to UDC11
FORTRAN Interface: Identical to UDC11

Functional Differences: None

14.7.2.4 Momentary Digital Output

MACRO Interface:

User interface is via the QIO IO.MSC issued to the ICS/ICR-11 driver. The UDC11 driver does not support this function since the module may be accessed directly through the UDC device common block.

FORTRAN Interface:

Identical to UDC11; however, existing FORTRAN tasks must be relinked to include ICS/ICR-11 FORTRAN interface routines.

Functional Differences:

Momentary output operations are now processed by the ICS/ICR-11 driver, rather than through direct access to the I/O page.

14.7.2.5 Noninterrupting Digital Input

MACRO Interface:

MACRO Interface is by means of the ICS/ICR-11 device common block and mapping table described in section 14.6.

FORTRAN Interface: Identical to UDC11; however, existing tasks must be relinked to include revised ICS/ICR-11 FORTRAN interface routines.

Functional Differences: None

14.7.2.6 Analog Output

MACRO Interface:

User interface is via the QIO IO.SAO issued to the ICS/ICR-11 driver. The UDC11 driver does not support this function since the module may be accessed directly through the UDC device common block.

FORTRAN Interface:

Identical to UDC11; however, existing FORTRAN tasks must be relinked to include ICS/ICR-11 FORTRAN interface subroutines.

Functional Differences:

Analog output operations are now processed by the ICS/ICR-11 driver rather than through direct access to the I/O page.

14.7.2.7 Interrupting Digital Input

MACRO Interface: Identical to UDC11 driver

FORTRAN Interface:

Identical to UDC11 driver; however, if digital inputs are read through the call to RCIPT then the task must be relinked to incorporate the revised ICS/ICR-11 FORTRAN interface routines.

Functional Differences: None.

APPENDIX A

SUMMARY OF I/O FUNCTIONS

This appendix summarizes legal I/O functions for all device drive described in this manual. Both devices and functions are list alphabetically. The meanings of the five parameters represented the ellipsis (...) are described in section 1.5.1. The meanings the function-specific parameters shown below are discussed in t appropriate driver chapters. The user may reference these functio symbolically by invoking the system macros FILIO$ (standard I functions) and SPCIO$ (special I/O functions), or by allowing them be defined at task build time from the system object library.

A.1 ANALOG-TO-DIGITAL CONVERTER DRIVERS

IO.KIL,...	Cancel I/O requests
IO.RBC,...,<stadd,size,stcnta>	Initiate an A/D conversion

A.2 CARD READER DRIVER

IO.ATT,...	Attach device
IO.DET,...	Detach device
IO.KIL,...	Cancel I/O requests
IO.RDB,...,<stadd,size>	Read logical block (binary)
IO.RLB,...,<stadd,size>	Read logical block (alphanumeric)
IO.RVB,...,<stadd,size>	Read virtual block (alphanumeric)

A.3 CASSETTE DRIVER

IO.ATT,...	Attach device
IO.DET,...	Detach device
IO.EOF,...	Write end-of-file gap
IO.KIL,...	Cancel I/O requests
IO.RLB,...,<stadd,size>	Read logical block
IO.RVB,...,<stadd,size>	Read virtual block

IO.RWD,...	Rewind tape
IO.SPB,...,<nbs>	Space blocks
IO.SPF,...,<nes>	Space files
IO.WLB,...,<stadd,size>	Write logical block
IO.WVB,...,<stadd,size>	Write virtual block

A.4 COMMUNICATION DRIVERS (MESSAGE-ORIENTED)

IO.ATT,...	Attach device
IO.DET,...	Detach device
IO.FDX,...	Set device to full duplex mode
IO.HDX,...	Set device to half-duplex mode
IO.INL,...	Initialize device and set device characteristics
IO.RLB,...,<stadd,size>	Read logical block, stripping sync characters
IO.RNS,...,<stadd,size>	Read logical block, transparent mode
IO.SYN,...,<syn>	Specify sync character
IO.TRM,...	Terminate communication, disconnecting from physical channel
IO.WLB,...,<stadd,size>	Write logical block with sync leader
IO.WNS,...,<stadd,size>	Write logical block, no sync leader

A.5 DECTAPE DRIVER

IO.RLB,...,<stadd,size,,,lbn>	Read logical block (forward)
IO.RLV,...,<stadd,size,,,lbn>	Read logical block (reverse)
IO.RVB,...,<stadd,size,,,lbn>	Read virtual block (forward)
IO.WLB,...,<stadd,size,,,lbn>	Write logical block (forward)
IO.WLV,...,<stadd,size,,,lbn>	Write logical block (reverse)
IO.WVB,...,<stadd,size,,,lbn>	Write virtual block (forward)

A.6 DISK DRIVER

IO.RLB,...,<stadd,size,,blkh,blkl>	Read logical block
IO.RPB,...,<stadd,size,,,pbn>	Read physical block

IO.RVB,...,<stadd,size,,blkh,blkl>	Read virtual block
IO.WDD,...,<stadd,size,,,pbn>	Write physical block (with deleted data mark)
IO.WLB,...,<stadd,size,,blkh,blkl>	Write logical block
IO.WPB,...,<stadd,size,,,pbn>	Write physical block
IO.WVB,...,<stadd,size,,blkh,blkl>	Write virtual block

A.7 INDUSTRIAL CONTROL LOCAL AND REMOTE SUBSYSTEMS

IO.CCI,...<stadd,sizb,tevf>	Connect a buffer to digital interrupts
IO.CTI,...<stadd,sizb,tevf,arv>	Connect a buffer to counter interrupts
IO.CTY,...<stadd,sizb,tevf>	Connect a buffer to terminal interrupts
IO.DCI,...	Disconnect a buffer from digital interrupts
IO.DTI,...	Disconnect a buffer from counter interrupts
IO.DTY,...	Disconnect a buffer from terminal interrupts
IO.FLN,...	Set controller offline
IO.ITI,...<mn,ic>	Initialize a counter
IO.LDI,...<tname,,[tevf],pn,csm>	Link task to digital interrupts
IO.LKE,...<tname,,[tevf]>	Link task to error interrupts
IO.LTI,...<tname,,[tevf],cn,[arv]>	Link task to counter interrupts
IO.LTY,...<tname,,[tevf]>	Link task to remote terminal interrupts
IO.MLO,...<opn,pp,dp>	Open or close bistable digital output points
IO.MSO,...<opn,dp>	Pulse single-shot digital output points
IO.NLK,...<tname>	Unlink a task from all interrupts
IO.NLN,...	Place ICR controller online
IO.RAD,...<stadd>	Read activating data
IO.RBC,...<stadd,size,stcnta>	Initiate multiple A/D conversions

IO.SAO,...<chn,vout>	Perform analog output
IO.UDI,...<tname>	Unlink a task from digital interrupts
IO.UER,...<tname>	Unlink a task from error interrupts
IO.UTI,...<tname>	Unlink a task from counter interrupts
IO.UTY,...<tname>	Unlink a task from terminal interrupts
IO.WLB,...<staddb,sizb>	Transmit data to the ICR remote terminal

A.8 LABORATORY PERIPHERAL SYSTEMS DRIVERS

IO.ADS,...,<stadd,size,pnt, ticks,bufs,chna>	Perform A/D sampling
IO.HIS,...,<stadd,size,pnt, ticks,bufs>	Perform histogram sampling
IO.KIL,...	Cancel I/O requests
IO.LED,...,<int,num>	Display number in LED lights
IO.MDA,...,<stadd,size,pnt, ticks,bufs,chnd>	Perform D/A output
IO.MDI,...,<stadd,size,pnt, ticks,bufs,mask>	Perform digital input sampling
IO.MDO,...,<stadd,size,pnt, ticks,bufs,mask>	Perform digital output
IO.REL,...,<rel,pol>	Latch output relay
IO.SDI,...,<mask>	Read digital input register
IO.SDO,...,<mask,data>	Write digital output register
IO.STP,...,<stadd>	Stop in-progress request

A.9 LINE PRINTER DRIVER

IO.ATT,...	Attach device
IO.DET,...	Detach device
IO.KIL,...	Cancel I/O requests
IO.WLB,...,<stadd,size,vfc>	Write logical block
IO.WVB,...,<stadd,size,vfc>	Write virtual block

A.10 MAGNETIC TAPE DRIVER

IO.ATT,...	Attach device
IO.DET,...	Detach device
IO.EOF,...	Write end-of-file (tape mark)
IO.KIL,...	Cancel I/O requests
IO.RLB,...,<stadd,size>	Read logical block
IO.RVB,...,<stadd,size>	Read virtual block
IO.RWD,...	Rewind tape
IO.RWU,...	Rewind and turn unit off-line
IO.SEC,...	Read tape characteristics
IO.SMO,...,<cb>	Mount tape and set tape characteristics
IO.SPB,...,<nbs>	Space blocks
IO.SPF,...,<nes>	Space files
IO.STC,...,<cb>	Set tape characteristics
IO.WLB,...,<stadd,size>	Write logical block
IO.WVB,...,<stadd,size>	Write virtual block

A.11 PAPER TAPE READER/PUNCH DRIVERS

IO.ATT,...	Attach device
IO.DET,...	Detach device
IO.KIL,...	Cancel I/O Requests
IO.RLB,...,<stadd,size>	Read logical block (reader only)
IO.RVB,...,<stadd,size>	Read virtual block (reader only)
IO.WLB,...,<stadd,size>	Write logical block (punch only)
IO.WVB,...,<stadd,size>	Write virtual block (punch only)

A.12 TERMINAL DRIVER

IO.ATT,...	Attach device
IO.DET,...	Detach device
IO.KIL,...	Cancel I/O requests
IO.RAL,...,<stadd,size>	Read logical block and pass all bits

IO.RLB,...,<stadd,size>	Read logical block
IO.RVB,...,<stadd,size>	Read virtual block
IO.WAL,...,<stadd,size>	Write logical block and pass all bits
IO.WLB,...,<stadd,size,vfc>	Write logical block
IO.WVB,...,<stadd,size,vfc>	Write virtual block

A.13 UNIVERSAL DIGITAL CONTROLLER DRIVER

IO.CCI,...,<stadd,sizb,tevf>	Connect a buffer to contact interrupts
IO.CTI,...,<stadd,sizb,tevf,arv>	Connect a buffer to timer interrupts
IO.DCI,...	Disconnect a buffer from contact interrupt
IO.DTI,...	Disconnect a buffer from timer interrupts
IO.ITI,...,<mn,ic>	Initialize a timer
IO.KIL,...	Cancel I/O requests
IO.MLO,...,<opn,pp,dp>	Open or close latching digital output points
IO.RBC,...,<stadd,size,stcnta>	Initiate multiple A/D conversions

APPENDIX B

I/O FUNCTION AND STATUS CODES

This appendix lists the numeric codes for all I/O functions, directive status returns, and I/O completion status returns. Lists are organized in the following sequence:

- I/O completion status codes
- Directive status codes
- Device-independent I/O function codes
- Device-dependent I/O function codes

Device-dependent function codes are listed by device. Both devices and codes are organized in alphabetical order.

For each code, the symbolic name is listed in form IO.xxx, IE.xxx, or IS.xxx. A brief description of the error or function is also included. Both decimal and octal values are provided for all codes.

B.1 I/O STATUS CODES

This section lists error and success codes which can be returned in the I/O status block on completion of an I/O function. The codes below may be referenced symbolically by invoking the system macro IOERR$.

B.1.1 I/O Status Error Codes

Name	Decimal	Octal	Meaning
IE.ABO	-15	177761	Operation aborted
IE.ALN	-34	177736	File already open
IE.BAD	-01	177777	Bad parameter
IE.BBE	-56	177710	Bad block
IE.BLK	-20	177754	Illegal block number
IE.BYT	-19	177755	Byte-ligned buffer specified
IE.CON	-22	177752	UDC connect error

I/O FUNCTION AND STATUS CODES

Name	Decimal	Octal	Meaning
IE.DAA	-08	177770	Device already attached
IE.DAO	-13	177763	Data overrun
IE.DNA	-07	177771	Device not attached
IE.DNR	-03	177775	Device not ready
IE.EOF	-10	177766	End-of-file encountered
IE.EOT	-62	177702	End-of-tape encountered
IE.EOV	-11	177765	End-of-volume encountered
IE.FHE	-59	177705	Fatal hardware error
IE.FLN	-81	177657	ICS/ICR controller already offline
IE.IFC	-2	177776	Illegal function
IE.MOD	-21	177753	Invalid UDC or ICS/ICR module
IE.NLK	-79	177661	Task not linked to specified ICS/ICR interrupts
IE.NLN	-37	177733	File not open
IE.NOD	-23	177751	No dynamic memory available to allocate a secondary control block
IE.NST	-80	177660	Task specified in ICS/ICR Link or Unlink request is not installed
IE.OFL	-65	177677	Device off-line
IE.ONP	-05	177773	Illegal subfunction
IE.OVR	-18	177756	Illegal read overlay request
IE.PRI	-16	177760	Privilege violation
IE.RSU	-17	177757	Nonsharable resource in use
IE.SPC	-06	177772	Illegal address space
IE.VER	-04	177774	Unrecoverable error
IE.WLK	-12	177764	Write-locked device

B.1.2 I/O Status Success Codes

Name	Decimal	Octal	Meaning
IS.CR	Byte 0: 1 Byte 1: 15	006401	Successful completion with carriage return
IS.ESC	Byte 0: 1 Byte 1: 33	015401	Successful completion with ESCape
IS.PND	+00	000000	I/O request pending
IS.RDD	+02	000002	Deleted data mark read
IS.SUC	+01	000001	Successful completion

B.2 DIRECTIVES CODES

This section lists error and success codes which can be returned in the directive status word at symbolic location $DSW when a QIO directive is issued.

B.2.1 Directive Error Codes

Name	Decimal	Octal	Meaning
IE.ADP	-98	177636	Invalid address
IE.IEF	-97	177637	Invalid event flag number
IE.ILU	-96	177640	Invalid logical unit number
IE.SDP	-99	177635	Invalid DIC number or DPB size
IE.ULN	-05	177773	Unassigned LUN
IE.UPN	-01	177777	Insufficient dynamic storage

B.2.2 Directive Success Codes

Name	Decimal	Octal	Meaning
IS.SUC	+01	000001	Directive accepted

B.3 I/O FUNCTION CODES

This section lists codes for all standard and device-dependent I/O functions.

B.3.1 Standard I/O Function Codes

Name	Octal Words	Octal Code	Bytes Subcode	Meaning
IO.ATT	001400	3	0	Attach device
IO.DET	002000	4	0	Detach device
IO.KIL	000012	0	12	Cancel I/O requests
IO.RLB	001000	2	0	Read logical block
IO.RVB	010400	21	0	Read virtual block
IO.WLB	000400	1	0	Write logical block
IO.WVB	011000	22	0	Write virtual block

B.3.2 Specific A/D Converter I/O Function Codes

Name	Octal Words	Octal Code	Bytes Subcode	Meaning
IO.RBC	003000	6	0	Initiate an A/D conversion

B.3.3 Specific Card Reader I/O Function Codes

Name	Octal Words	Octal Code	Bytes Subcode	Meaning
IO.RDB	001200	2	200	Read logical block (binary)

B.3.4 Specific Cassette I/O Function Codes

Name	Octal Words	Octal Code	Bytes Subcode	Meaning
IO.EOF	003000	6	0	Write end-of-file gap
IO.RWD	002400	5	0	Rewind tape
IO.SPB	002420	5	20	Space blocks
IO.SPF	002440	5	40	Space files

B.3.5 Specific Communication (Message-Oriented) I/O Function Codes

Name	Octal Words	Octal Code	Bytes Subcode	Meaning
IO.FDX	003020	6	20	Set device to full-duplex mode
IO.HDX	003010	6	10	Set device to half-duplex mode
IO.INL	002400	5	0	Initialize device and set device characteristics
IO.RNS	001020	2	20	Read logical block, transparent mode
IO.SYN	003040	6	40	Specify sync character
IO.TRM	002310	5	10	Terminate communication, disconnecting from physical channel
IO.WNS	000420	1	20	Write logical block with no sync leader

B.3.6 Specific DECtape I/O Function Codes

Name	Octal Words	Octal Code	Bytes Subcode	Meaning
IO.RLV	001100	2	100	Read logical block (reverse)
IO.WLV	000500	1	100	Write logical block (reverse)

B.3.7 Specific Disk I/O Function Codes (RX01)

Name	Octal Words	Octal Code	Bytes Subcode	Meaning
IO.RPB	001040	2	40	Read physical block
IO.WPB	000440	1	40	Write physical block

B.3.8 Specific ICS/ICR I/O Function Codes

Name	Octal Words	Octal Code	Bytes Subcode	Meaning
IO.CCI	014000	30	0	Connect a buffer to digital interrupt input
IO.CTI	015400	33	0	Connect a counter
IO.CTY	003400	7	0	Connect a remote terminal

IO.DCI	014400	31	0	Disconnect a buffer from digital interrupt input
IO.DTI	016000	34	0	Disconnect a buffer from counter input
IO.DTY	006400	15	0	Disconnect a buffer from terminal input
IO.FLN	012400	25	0	Place selected unit offline
IO.ITI	017000	36	0	Initialize a counter
IO.LDI	007000	16	0	Link a task to digital interrupts
IO.LKE	012000	24	0	Link a task to error interrupts
IO.LTI	007400	17	0	Link a task to counter interrupts
IO.LTY	010000	20	0	Link a task to terminal interrupts
IO.MLO	006000	14	0	Open or close bistable digital output points
IO.MSO	005000	12	0	Pulse single-shot digital output points
IO.NLK	011400	23	0	Unlink a task from all unsolicited interrupts
IO.NLN	017400	37	0	Place selected unit online
IO.RAD	010400	21	0	Read task activation data
IO.RBC	003000	6	0	Initiate multiple A/D conversions
IO.SAO	004000	10	0	Perform analog output to specified channel
IO.UDI	011410	23	10	Unlink a task from digital interrupts
IO.UER	011440	23	40	Unlink a task from error interrupts
IO.UTI	011420	23	20	Unlink a task from counter interrupts
IO.UTY	011430	23	30	Unlink a task from terminal interrupts
IO.WLB	000400	1	0	Output to remote terminal

B.3.9 Specific LPS I/O Function Codes

Name	Octal Words	Octal Code	Bytes Subcode	Meaning
IO.ADS	014000	30	0	Initialize A/D sampling
IO.HIS	015000	32	0	Initialize histogram sampling
IO.LED	012000	24	0	Display number in LED lights
IO.MDA	016000	34	0	Initialize D/A output
IO.MDI	014400	31	0	Initialize digital input sampling
IO.MDO	015400	33	0	Initialize digital output
IO.REL	013400	27	0	Latch output relay
IO.SDI	013000	26	0	Read digital input register
IO.SDO	012400	25	0	Write digital output register
IO.STP	016400	35	0	Stop in-progress request

B.3.10 Specific Magtape I/O Function Codes

Name	Octal Words	Octal Code	Bytes Subcode	Meaning
IO.EOF	003000	6	0	Write end-of-file gap
IO.RWD	002400	5	0	Rewind tape
IO.RWU	002540	5	140	Rewind and unload
IO.SEC	002520	5	120	Sense characteristics
IO.SMO	002560	5	160	Mount and set characteristics
IO.SPB	002420	5	20	Space blocks
IO.SPF	002440	5	40	Space files
IO.STC	002500	5	100	Set characteristics

B.3.11 Specific Terminal I/O Function Codes

Name	Octal Words	Octal Code	Bytes Subcode	Meaning
IO.RAL	001010	2	10	Read pass all bits
IO.WAL	000410	1	10	Write pass all bits

B.3.12 Specific UDC I/O Function Codes

Name	Octal Words	Octal Bytes Code	Subcode	Meaning
IO.CCI	014000	30	0	Connect a buffer to contact interrupt digital input
IO.CTI	015400	33	0	Connect a timer
IO.DCI	014400	31	0	Disconnect a buffer from contact interrupt digital input
IO.DTI	016000	34	0	Disconnect a timer
IO.ITI	017000	36	0	Initialize a timer
IO.MLO	006000	14	0	Open or close latching digital output points
IO.RBC	003000	6	0	Initiate multiple A/D conversions

APPENDIX C

RSX-11M PROGRAMMING EXAMPLE

```
MESSAGE TRANSFER        MACRO M0710  10-OCT-74 10:19  PAGE 1


                        .TITLE MESSAGE TRANSFER
                        .IDENT  /01/

                ;
                ; COPYRIGHT   1974,  DIGITAL  EQUIPMENT  CORP.,  MAYNARD,   MASS.
                ;
                ; THIS SOFTWARE IS FURNISHED TO PURCHASER UNDER A LICENSE FOR USE
                ; ON A  SINGLE COMPUTER SYSTEM AND CAN BE  COPIED (WITH INCLUSION
                ; OF DEC'S COPYRIGHT  NOTICE) ONLY FOR USE IN SUCH SYSTEM, EXCEPT
                ; AS MAY OTHERWISE BE PROVIDED IN WRITING BY DEC.
                ;
                ; THE  INFORMATION  IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
                ; NOTICE  AND  SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
                ; EQUIPMENT CORPORATION.
                ;
                ; DEC  ASSUMES NO  RESPONSIBILITY  FOR  THE  USE  OR  RELIABILITY
                ; OF  ITS  SOFTWARE ON  EQUIPMENT WHICH  IS  NOT SUPPLIED BY DEC.
                ;
                ; VERSION 01
                ;
                ; EARL WALDIN   5-SEP-74
                ;
                ; DEMONSTRATION OF USE OF RSX-11M I/O
                ;
                ; MACRO LIBRARY CALLS
                ;

                        .MCALL  ALUN$S,QIO$S,WTSE$S,WSIG$S


                ;
                ; LOCALLY DEFINED MACROS
                ;

                        .MACRO  CALL SUBR       ;DEFINITION FOR SUBROUTINE CALLS
                        JSR     PC,SUBR
                        .ENDM

                        .MACRO  RETURN          ;SUBROUTINE RETURN MACRO
                        RTS     PC
                        .ENDM

                ;
                ; LOCAL DATA
                ;


                ;
```

```
 47                     ; READ-RELATED STORAGE
 48                     ;
 49
 50 000000  000000  RDSTS:  .WORD   0               ;READ STATUS BLOCK
 51 000002  000000          .WORD   0
 52
 53                     ;
 54                     ; WRITE-RELATED STORAGE
 55                     ;
 56
 57 000004  000000  WRSTS:  .WORD   0               ;WRITE STATUS BLOCK


MESSAGE TRANSFER        MACRO M0710  10-OCT-74 10:19  PAGE 1-1


 58 000006  000000          .WORD   0
 59
 60                     ;
 61                     ; BUFFER STORAGE
 62                     ;
 63
 64 000010          BUF1:   .BLKB   82.             ;BUFFER 1
 65 000132          BUF2:   .BLKB   82.             ;BUFFER 2
 66
 67                     ;+
 68                     ; **-$XFER-DEMONSTRATE USE OF RSX-11M I/O BY OUTPUTTING RECORDS
 69                     ; FROM TI: (USER'S TERMINAL) TO LINE PRINTER. REQUESTS ARE DOUBLE
 70                     ; BUFFERED TO DEMONSTRATE HOW OPERATIONS MAY BE OVERLAPPED.
 71                     ;-
 72
 73 000254          $XFER:: ALUN$S  #1,#"TI,#0      ;LUN 1 IS TI: DEVICE
 74 000274                  ALUN$S  #2,#"LP         ;LUN 2 IS LP0:
 75                     ;
 76                     ; READ A LINE FROM INPUT DEVICE, LUN 1
 77                     ;
 78 000314          10$:    QIO$S   #IO.RLB,#1,#1,,#RDSTS,,<#BUF1,#80.>
 79 000366  103003          BCC     20$             ;IF DISPATCHED OK, CONTINUE
 80 000370                  CALL    STCHK           ;CHECK STATUS
 81 000374  000747          BR      10$             ;IF RECOVERABLE ERROR, TRY AGAIN
 82
 83 000376          20$:    WTSE$S  #1              ;WAIT UNTIL 1 COMPLETE
 84 000410  126727          CMPB    RDSTS,#IS.SUC   ;READ SUCCESSFUL?
            177364
            000000G
 85 000416  001402          BEQ     30$             ;CONTINUE IF SUCCESSFUL
 86 000420  000167          JMP     100$            ;TERMINATE IF NOT SUCCESSFUL
            000440
 87
 88 000424  016701  30$:    MOV     RDSTS+2,R1      ;GET ACTUAL BYTE COUNT IN R1
            177352
 89                     ;
 90                     ; BEGIN TO FILL SECOND BUFFER
 91                     ;
 92 000430          40$:    QIO$S   #IO.RLB,#1,#2,,#RDSTS,,<#BUF2,#80.>
 93 000502  103003          BCC     50$             ;CONTINUE IF DISPATCH OK
 94 000504                  CALL    STCHK           ;CHECK STATUS
 95 000510  000747          BR      40$             ;TRY AGAIN
 96
 97                     ;
 98                     ; START BUFFER 1 OUT
 99                     ;
100
101 000512          50$:    QIO$S   #IO.WLB,#2,#1,,#WRSTS,,<#BUF1,R1,#40>
102 000564  103003          BCC     60$             ;CONTINUE IF NO DISPATCH ERROR
103 000566                  CALL    STCHK           ;CHECK STATUS
104 000572  000747          BR      50$             ;TRY AGAIN
105
106                    ;
107                    ; THIS IS A SYNCHRONIZATION POINT. BOTH FUNCTIONS MUST COMPLETE
108                    ; BEFORE ANYTHING ELSE BEGINS.
109                    ;
110
```

```
MESSAGE TRANSFER        MACRO M0710  10-OCT-74 10:19  PAGE 1-2


    111 000574          60$:    WTSE$S  #2              ;WAIT FOR 2 TO FILL
    112 000606  126727          CMPB    RDSTS,#IS.SUC   ;SUCCESSFUL?
                177166
                000000G
    113 000614  001123          BNE     100$            ;IF NOT, CRASH
    114 000616  016702          MOV     RDSTS+2,R2      ;GET COUNT FOR BUFFER 2
                177160
    115 000622                  WTSE$S  #1              ;WAIT FOR 1 TO EMPTY
    116 000634  126727          CMPB    WRSTS,#IS.SUC   ;SUCCESSFUL?
                177144
                000000G
    117 000642  001110          BNE     100$            ;IF NOT, CRASH
    118                 ;
    119                 ; FILL BUFFER 1, EMPTY BUFFER 2
    120                 ;
    121 000644          70$:    QIO$S   #IO.RLB,#1,#1,,#RDSTS,,<#BUF1,#80.>
    122 000716  103003          BCC     80$             ;IF OK, CONTINUE
    123 000720                  CALL    STCHK           ; CHECK STATUS
    124 000724  000747          BR      70$             ;TRY AGAIN
    125
    126 000726          80$:    QIO$S   #IO.WLB,#2,#2,,#WRSTS,,<#BUF2,R2,#40>
    127 001000  103003          BCC     90$             ;CONTINUE IF SUCCESSFUL
    128 001002                  CALL    STCHK           ;CHECK STATUS IF NOT SUCCESSFUL
    129 001006  000747          BR      80$             ;RETURN
    130
    131                 ;
    132                 ; THIS IS ALSO A SYNCHRONIZATION POINT
    133                 ;
    134
    135 001010          90$:    WTSE$S  #1              ;WAIT FOR 1 TO FILL
    136 001022  126727          CMPB    RDSTS,#IS.SUC   ;SUCCESSFUL?
                176752
                000000G
    137 001030  001015          BNE     100$            ;IF NOT, CRASH
    138 001032  016701          MOV     RDSTS+2,R1      ;GET ACTUAL BYTE COUNT IN R1
                176744
    139 001036                  WTSE$S  #2              ;WAIT FOR BUFFER 2 TO EMPTY
    140 001050  126727          CMPB    WRSTS,#IS.SUC   ;SUCCESSFUL?
                176730
                000000G
    141 001056  001002          BNE     100$            ;TERMINATE IF NOT
    142 001060  000167          JMP     40$             ;BACK INTO LOOP
                177344
    143
    144                 ;
    145                 ; DON'T ATTEMPT TO RECOVER ERRORS
    146                 ;
    147
    148 001064  000004  100$:   IOT                     ;CRASH TASK
    149
    150                 ;+
    151                 ; **- STCHK - ATTEMPT TO RECOVER DIRECTIVE DISPATCH ERROR ONLY IF
    152                 ; IT INVOLVES DYNAMIC MEMORY ALLOCATION - OTHERWISE TERMINATE.
    153                 ;
    154                 ;       INPUTS:
    155                 ;
    156                 ;               (SP)=RETURN ADDRESS
```

```
MESSAGE TRANSFER        MACRO M0710  10-OCT-74 10:19  PAGE 1-3


    157                 ;
    158                 ;       OUTPUTS:
    159                 ;
    160                 ;               NONE
    161                 ;-
    162
    163 001066  126727  STCHK:  CMPB    $DSW,#IE.UPN    ;BUFFER ALLOCATION FAILURE?
                000000G
                000000G
    164 001074  001004          BNE     10$             ;IF NOT TERMINATE
    165 001076                  WSIG$S                  ;AWAIT SIGNIFICANT EVENT
    166 001104                  RETURN                  ;TRY AGAIN
    167
    168 001106  000004  10$:    IOT                     ;CRASH TASK
    169
    170         000254'         .END    $XFER
```

```
MESSAGE TRANSFER        MACRO M0710  10-OCT-74 10:19  PAGE 1-4
SYMBOL TABLE

BJF1    000010R         IO.WLB= ****** GX       WRSTS   000004R
BJF2    000132R         IS.SUC= ****** GX       $DSW  = ****** GX
IE.JPN= ****** GX       RDSTS   000000R         $XFER   000254RG
IO.RLB= ****** GX       STCHK   001066R         $$$ARG= 000002

. ABS.  000000      000
        001110      001
ERRORS DETECTED:  0

FREE CORE:  3586. WORDS
,MSG/LI:TTM=MSG.001
```

APPENDIX D

GLOSSARY OF RSX-11M TERMS

ASYNCHRONOUS SYSTEM TRAP (AST)

A system condition which occurs as a result of an external significant event such as completion of an I/O request. On occurrence of the significant event, control passes to an AST service routine, and the AST is added to an Executive first-in first-out queue for the task in which the service routine appears.

ATTACH

Dedicate a physical device unit for exclusive use by the task that requested attachment. Once the physical device unit has been attached by a task, using an IO.ATT I/O function, only that task can free the unit again for use by other tasks in the system. Attachment request attempted to a device unit already attached by another task will not be terminated until the attachment request can finally be honored; in other words, the attachment request is terminated only when the previous attachment is terminated, and no higher priority attachment requests are queued.

DETACH

Free an attached physical device unit for use by tasks other than the one that attached it. A physical device unit can only be detached, by means of an IO.DET I/O function, by the task that attached it, or by the Executive if the task is terminated with the device still attached.

DIRECTIVE

A type of system meta-instruction which is used to provide a facility inherent in the hardware by means of executive requests issued to the RSX-11M Executive. Directives are usually invoked by means of execution of expanded code from macros in the System Macro Library (RSXMAC.SML).

EVENT FLAG NUMBER

A number which can be specified in a QIO or other macro call to indicate to the issuing task which significant event has

occurred. There are 64 event flags available in RSX-11M. Flags numbered 1 through 32 are local to a task; 33 through 64 are common to all tasks. Flags 25 through 32 and 57 through 64 are normally reserved for RSX-11M system software use. Each of the available flags can be referenced by number and can be used for communication and synchronization between user tasks, or between tasks and executive service requests, including I/O requests.

I/O STATUS BLOCK

A 2-word array (double-word) in which a code representing the final status of an I/O request is returned, if the address of the block is specified in the QIO directive parameter block (DPB) which generated the request. A code identifying the type of success or error is returned in the low-order byte of the first word, optional device-dependent information in the high-order byte of the first word, and the number of bytes transferred on a read or write in the second word of the block. Although the I/O Status Block is optional, it is the only way a user can guarantee that he will know the outcome of an I/O request.

LOGICAL ADDRESS

A logical address is a software representation of a hardware address. The use of the phrase "logical address" implies that some mapping occurs between "true", or hardware address and "artificial" or logical address. The reason for using logical addresses is that they simplify the way one deals with a hardware device or family of devices.

The logical address in RSX-11M refers to the relative position of a logical block on a volume. The volume is divided into logical blocks, each of which is assigned an address called a logical block number (LBN). All mass storage media are accessed by LBN (e.g., 17) rather than physical address (e.g., cylinder 5, track 3, sector 7).

LOGICAL BLOCK

A logical block in RSX-11M parlance refers to 512 bytes of storage which may be considered to be a discrete entity for logical purposes. In fact, it might be composed of odd-sized fragments of non-contiguous storage. Actually, a logical block generally refers to one or more physical blocks of a formatted or block-structured mass memory which compose the logical atom for access to the medium. Logical block may also refer to the in-core image of a logical block which is or will be on a mass storage device.

The concept of logical block is useful on file-structured devices, in that all such devices appear to share all these characteristics except total number of blocks.

LOGICAL BLOCK NUMBER (LBN)

Sequential position of a logical block with respect to a collection of such blocks (which may compose a volume). If the collection of blocks had been written in logical order on a sequential medium, such as magnetic tape, the logical block number for any block would be the true position of the block on that medium, e.g., logical block 2 would be encountered just after LBN 1 and just before LBN 3.

LOGICAL UNIT NUMBER (LUN)

A number associated with a physical device unit during a task's I/O operations. Each task in the system can establish its own correspondence between LUNs and physical device units.

MACRO

A system capability which allows a user to generate Assembler instructions, data, or symbols in a predetermined format by providing actual arguments to the Assembler in a macro call included in a MACRO-11 program. Macros provide a standardized means of obtaining access to system services or resources by invocations from programs.

PRIORITY

A number associated with an RSX-11M task which indicates the relative position of that task among all tasks in the system. The priority is associated with a task at task build time and may be changed at install or run time. Legal priorities are in range 1 through 250, with greater magnitude indicating higher priority. If two tasks are identical in every way (i.e., resources used, etc.) except priority and are initiated at the same time, the task with the higher priority will complete first. I/O requests issued by a task assume the priority of that task and are honored according to the task's priority.

SIGNIFICANT EVENT

An event or condition which indicates a change in system status. In RSX-11M, a significant event is declared when an I/O operation completes and in some other cases as well. A declaration of a significant event indicates that the Executive should review the eligibility of all tasks in the system to determine which task should run next, since the significant event might unblock the execution of a higher priority task.

SYNCHRONOUS SYSTEM TRAP (SST)

A system condition which occurs as a result of an error or fault within the executing task. If the same instruction sequence were repeated, the same synchronous trap would occur. On recognition of a synchronous trap, control passes to an SST service routine. SSTs are not handled directly by the Executive as ASTs are.

VIRTUAL ADDRESS

A number which indicates relative position within a collection of logically-related granules of a storage medium. The fact that the medium itself may be virtual (e.g., 1 million bytes of addressable memory, but only 64K in core memory, the remainder on mass storage) is of little consequence; in fact, the ability to deal with a hierarchical or multi-level memory as if it were one medium is one of the principal advantages of systems supporting virtual addressing. In RSX-11M, virtual address generally refers to relative position within a task image, while VIRTUAL BLOCK NUMBER (VBN) refers to relative position within a file.

VIRTUAL BLOCK

One of a collection of blocks which make up a user file (or the core image of that file). The block is virtual only in that its address (VBN) refers to position within a file regardless of the file's allocation or placement on a storage medium. When a user accesses a file, he can think of the file as a virtual storage medium belonging to him. Virtual addressing within that file could be considered to be absolute addressing on a virtual medium.

INDEX

RSX-11M I/O Drivers
Reference Manual
DEC-11-OMDRA-B-D

READER'S COMMENTS

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

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

_____

_____

_____

_____

_____

_____

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

_____

_____

_____

_____

_____

_____

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

_____

_____

_____

_____

_____

_____

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

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

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

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

Please cut along this line.

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

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

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

)ESS REPLY MAIL
)STAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

e will be paid by:

digital

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

digital equipment corporation

Printed in U.S.A.