IAS Device Handlers Reference Manual

Order Number: AA–H004B–TC

This manual describes the use and characteristics of IAS device handlers.

Operating System and Version: IAS Version 3.4

May 1990

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

This document was prepared using VAX DOCUMENT, Version 1.2

Contents

EXAMPLES

FIGURES

TABLES

Preface

Manual Objectives and Reader Assumptions

The *IAS Device Handlers Reference Manual* provides a reference source for users of the device handler tasks that service the peripheral devices supported by Digital. You should be familiar with PDP-11 assembler language and with the appropriate user's guide.

Structure of the Document

Chapter 1 describes most of the characteristics common to each handler task. The remaining chapters describe either an individual handler task or a set of closely related handler tasks. Appendix A is a list of QIOMAC.MAC, the macro that defines queue I/O directive function values and status return values. If you want more information about writing a device handler, consult the *IAS Guide to Writing a Device Handler Task*.

Associated Documents

Documents that provide related information are described in the *IAS Master Index and Documentation Directory*.

1 Introduction

1.1 Device Handler Tasks

IAS provides a flexible, device-independent, and function-independent I/O capability that can support standard PDP-11 peripherals and special purpose devices. Peripheral device support is provided by privileged device handler tasks and is not an integral part of the Executive. Device handler tasks can be developed with a minimum knowledge of the Executive code.

Device handlers must be installed with task names of dd...., where dd corresponds to the two letter mnemonic of the device(s) that the handler services. The tasks must be resident and initialized before they can be used. This is effected using either of the following commands:

```
MCR>LOA
  or
PDS> RUN/HANDLER
```

See the *IAS MCR User's Guide* or the *IAS PDS User's Guide* for further details.

1.2 QIO System Directives

User tasks make I/O requests to device handlers by issuing a QIO system directive. System directives are discribed in the IAS System Directives Reference Manual. The arguments of the system directives determine the following:

1 The type of I/O desired via the function code specified,

2 The physical device on which the I/O request is to be performed via the logical unit number (LUN) argument, see Section 1.4,

3 The importance of the I/O service (via the priority of the request),

4 The execution mode of the I/O request relative to the user task: the request is performed either synchronously or asynchronously with the issuing user task. Execution mode is indicated by event flag and asynchronous system trap (AST) arguments.

1.3 Handler Task/User Task Interaction

When a standard QIO directive macro is specified in a user's program, the MACRO-11 assembler generates a directive parameter block (DPB) that holds the appropriate values, or generates code that pushes the DPB onto the stack at run time.

When a user task executes a QIO directive, the Executive takes the arguments from the DPB and creates an I/O request node in system common space. The system queues the request node (adds to a priority structured list of such nodes) to the device handler specified for service. When the user task's request node is the highest priority node capable of service, the handler task that services that device dequeues and processes the I/O request specified.

I/O requests are completed only if the DPB contains the proper arguments. After the device handler completes an I/O request, the Executive performs one or more of the following actions for the user task depending on the arguments in the QIO DPB.

1 Declares a significant event and sets a specified event flag. These functions allow the user program to perform synchronous I/O operations in the following manner:

   a. Issue a QIO system directive specifying an event flag (this immediately clears the event flag).

   b. Optionally execute some code within the user program.

   c. Issue a WAITFOR or STOPFOR system directive specifying the same event flag. This suspends the user program until completion of its I/O (allowing lower priority tasks to run). STOPFOR should be used if the I/O request may take a significant amount of time (a second or more), and the task can safely be checkpointed or swapped during this time.

      Alternatively, the user program can issue a QIO AND WAITFOR (QIOW$) system directive specifing an event flag.

      The QIOW$ is to be preferred when the task is waiting only for an I/O request which should complete quickly. The QIOW$ both reduces the number of directives performed and enables the executive to know why the task is waiting.

2 Saves current user task status, declares an asynchronous system trap (AST), and starts the user task at the AST address specified in the DPB. These functions allow the user program to perform asynchronous I/O operations in the following manner:

   a. Issue a QIO directive specifying the starting address of the AST service routine within the user task.

   b. Execute other instructions (including any further QIO directives).

   c. Execute its AST code transparently to the user's normal code when the I/O is completed (similar to an interrupt service routine). This feature permits user task multi-I/O streams to occur in parallel with the user task's execution.

3 Returns the status of the I/O operation from the device handler to a 2-word user status buffer defined in the DPB. This status code enables the user task to monitor the success or failure of its I/O. The status buffer format is as follows.

   - word1 - Byte 0 = I/O status code (see Appendix A, Byte 1 = 0 (normally unused)
   - word2 - For transfer requests, word2 holds the total number of bytes involved in the transfer.
   - For other requests, some handlers use this word to return status information. See the descriptions of individual handlers.

1.4 Specifying the Physical Device

Logical unit numbers have no connection to physical devices until the programmer or operator makes device assignments for a particular task. Device assignments tell the system that, for example, logical unit number 1 for user task A is associated with DECtape unit 3.

The system makes a correspondence between physical devices and logical unit numbers by means of a logical unit table (LUT) in the task's header. The LUT contains a user-specified number of entries, each of which corresponds to a logical unit number. Each entry contains a pointer to the Physical Unit Directory (PUD) entry for the device last assigned to that LUN. When a task issues

a QIO directive for a specified LUN, the system locates the physical device using the appropriate LUT entry. For example the physical device currently assigned to LUN 2 is identified using the second LUT entry.

Each user task has a set of logical unit assignments that can be created or altered in the following ways (alteration of logical unit assignments within one user task does not affect any other user task):

1 Using the MCR REASSIGN function or the DCL ASSIGN command, see the *IAS MCR User's Guide* and the *IAS PDS User's Guide.*

2 At user task build (link) time via the ASG option.

3 At run time via the ASSIGN LUN system directive issued by the program.

In the first two cases, the user task is unaware of the physical devices that correspond to its logical units. The task issues QIO system directives, specifying appropriate LUNs, while the actual I/O takes place interchangeably on a wide variety of system peripherals.

In the third case, the user task is aware of its physical device assignments, but not of any redirection done to the device.

1.5 QIO Macros

The QIO system directive is usually issued in the form of a system defined macro with fixed argument fields.

The forms of IAS directive macros are fully described in Section 1.5.1 of the IAS System Directives Reference Manual. The format of the QIO and QIOW macros is described in Chapter 4 of the same manual.

1.6 Function Codes (Non-Mass Storage)

The full range of global function codes available to a user task is specified in Appendix A. Seven function codes are common to most I/O operations and to almost every non-mass storage device. This basic subset of function codes is described in the following sections.

1.6.1 Attach/Detach

```
QIO$ IO.ATT,lun,ef,pri,iosb,ast ; ATTACH
QIO$ IO.DET,lun,ef,pri,iosb,ast ; DETACH
```

In a real-time or multi-user system, attach and detach I/O requests permit an eligible task to gain and release exclusive use of a peripheral device. These functions enable input and output to be processed in an unbroken stream; therefore, they are especially useful on sequential devices (non-file-structured devices; for example, terminal, line printer, card reader, paper tape). Attach causes a device to be dedicated to the task that issued the attach; Detach releases the device for use by other task.

The only tasks that can "break through" an attach by a user task are those running under a UIC of the form [1,n] through [7,n]. Such a UIC is called a "system UIC". A system UIC can have its requests dequeued, but it cannot take over the attach.

To attach a device, the QIO request is issued with a function code of IO.ATT. The attach remains in effect until a detach request code IO.DET is issued by the same task, specifying the LUN associated with the attach. While an attach is in effect, the I/O handler for the specified device dequeues only QIO directives issued either by the task that issued the attach or by tasks with a UIC of [1,n] through [7,n]. Should the task be aborted or exit before doing the detach, the Executive automatically detaches the device.

The attach/detach facility provides an automatic queueing mechanism for exclusive access to a device. If one task attempts to gain exclusive access to a device while it is attached to another task, the attach request will remain in the queue for the device until the currently attached task detaches. Thus several tasks may have attach requests in the queue at once, and exclusive access to the device will be granted to each in turn. However, this will not work for tasks running under a system UIC, because attach requests issued by such tasks will be dequeued immediately and then rejected by the handler, with a status of IE.DAA (device already attached).

In a timesharing system, devices can be attached in the same manner as described above, but by real-time tasks only. Further, a device to be attached must not be among those made available to timesharing users at timesharing start up.

In a timesharing system a device that is available for timesharing can be allocated to an individual terminal by the PDS> ALLOCATE command (see the *IAS PDS User's Guide*). Allocation gives the terminal the exclusive use of the device.

Real-time exclusive use and timesharing exclusive use of devices should be separated as far as possible. However this is not practical in many cases. Timesharing tasks can use the attach mechanism if the device is not already attached and not already allocated, or if the device is already allocated to the terminal for which the task is running. If the device is already attached to another task or allocated to another terminal, the attach request will remain in the queue for the device, as previously described.

When a terminal is attached, the opportunity can be taken:

1 In a real-time or multi-user system, to specify a task's response to CT/C, in place of a return to MCR or PDS.

2 In any type of system, to provide a response to unsolicited input at the terminal.

In these cases the function code IO.ATA is used in place of IO.ATT and either or both of two further AST entry point parameters are supplied in the QIO call. See Chapter 2, Sections "IO.ATA" and "IO.DET".

1.6.2 Read Logical/Read Virtual Block

```
QIO$ IO.RLB,lun,ef,pri,iosb,ast,<stadd,size,p3> ; Read Logical
QIO$ IO.RVB,lun,ef,pri,iosb,ast,<stadd,size,p3> ; Read Virtual
```

where:

Parameter	Meaning
stadd	Virtual starting address of the user's buffer for data input.
size	Size of the data buffer in bytes.
p3	Optional parameter(s) to specify special read modes or further arguments for certain devices.

The read logical block function reads an absolute block from a device, while read virtual block reads a relative block within a file. On a sequential device like the terminal or card reader, there is no difference in the functions.

1.6.3 Write Logical/Write Virtual Block

```
QIO$ IO.WLB,lun,ef,pri,iosb,ast,<stadd,size,p3> ;Write Logical
QIO$ IO.WVB,lun,ef,pri,iosb,ast,<stadd,size,p3> ;Write Virtual
```

where:

Parameter	Meaning
stadd	Virtual starting address of the user's buffer for data output.
size	Size of the data buffer in bytes.
p3	Optional parameter(s) to specify special write modes or further arguments for certain devices.

The write logical block function writes an absolute block to a device, while write virtual block writes a relative block within a file. On a sequential device like the terminal or line printer, there is no difference in the functions.

It is suggested that the write virtual block function (IO.WVB) be used for writes to all non-file-oriented devices because the system performs an access check under IO.WVB, but not in the case of IO.WLB. With IO.WLB a write can destroy the contents of a disk if output is accidentally directed to the wrong device by an executive privileged task.

1.6.4 Cancel (KILL I/O)

```
QIO$ IO.KIL,lun,ef,pri,iost,ast
```

The IO.KIL function is issued in special cases where a user task cancels all of its requests (pending, active and attach) for a particular device. This function is useful in releasing devices from which responses are overdue.

1.7 Function codes for Mass Storage Devices

Mass storage devices (that is, DECtape, magnetic tape and disks) are used in two modes of operation: Files-11 and direct. The use of Files-11 in a user task is described in the *IAS I/O Operations Reference Manual*. The following sections describe the use of the direct mode in a user task.

Use of direct mode I/O on a device that has a Files-11 volume mounted can result in destruction of information or corruption of the Files-11 directories on that volume.

1.7.1 Direct Mode

Direct mode operation of a mass storage device is used for either of the following situations:

1 The mass storage device has a non-Files-11 format (for example, DOS format handled by FILEX),

2 The mass storage device is unformatted (for example, device used to dump data acquired by an A/D converter).

When the MOUNT command (described below) is used to mount the device as FOREIGN, a user task may perform I/O operations directly to any logical block on the device. The concept of a virtual block no longer exists, since the system does not recognize the existence of files on the device. When this mode of operation is entered, the I/O functions described in Section 1.7.3 and Section 1.7.4 are available for use by a task.

For real-time or multi-user systems, any task has this access to a volume mounted as FOREIGN. In timesharing systems, a user who mounts a volume as FOREIGN gains sole access.

1.7.2 Mounting for Direct Mode

In both of the cases mentioned above, the mass storage unit must be mounted as a foreign volume before use, and dismounted after use by employing one of the following sets of commands. See also the *IAS PDS User's Guide* and the *IAS MCR User's Guide*.

DCL Commands

To mount a volume:

```
PDS> MOUNT/FOREIGN xxn: volume
```

where:

- MOUNT - Is the DCL MOUNT command
- /FOREIGN - Specifies that the volume is foreign and does not have Files-11 file structure
- xx - Is the device name (for example, DT, DK,)
- n - Is the device unit number in the range 0 through 7
- volume - Is the volume identification

To dismount a volume:

```
PDS> DISMOUNT xxn: volume
```

where:

- DISMOUNT - Is the PDS DISMOUNT command
- xx - Is the device name (for example DT,DK,)
- n - Is the device unit number (0-7)
- volume - Is the volume identification

MCR Commands

To mount a volume:

```
MCR>MOU xxn:/CHA=[FOR]
```

where:

- MOU - Is the MCR MOUNT command
- xx - Is the Device name (for example DT, DK)

- n - Is the Device unit number (0-7)
- CHA - Is the characteristics option
- [FOR] - Specifies that the volume is foreign and does not have Files-11 - file structure. The brackets are mandatory.

To dismount a volume:

```
MCR>DMO xxn:
```

where:

- DMO - Is the MCR DISMOUNT Command
- xx - Is the Device name (for example DT, DK)
- n - Is the Device unit number (0-7)

1.7.3 Attach/Detach

These functions are identical to the functions described in Section 1.6.1.

1.7.4 Read/Write Logical Block

I/O requests for mass storage devices in direct mode are issued via the QIO$ system macros whose formats are:

```
QIO$    IO.RLB,lun,ef,pri,iosb,ast,<stadd,size,comp,blkh,blkl>
QIO$    IO.WLB,lun,ef,pri,iosb,ast,<stadd,size,comp,blkh,blkl>
```

where:

Parameter	Meaning
stadd	Virtual starting address of user's buffer for data input or output. (This parameter must be on a word boundary and in some cases (for example RP03) an even word boundary.)
size	Size of the data buffer in bytes. The size must be even. For some peripherals it must also be a multiple of 4 bytes, or of 1000 bytes (256 words).
comp	0 (retains compatibility with non-mass storage logical read/write functions)
blkh, blkl	Block-high, block-low. Double precision number indicating the first logical block on the mass storage device on which the transfer is to take place; this forces block structure on word-oriented mass storage devices. The maximum value of each parameter depends upon the mass storage capacity of the unit.

1.7.5 Compatibility

Direct mode operation of mass storage devices promotes system compatibility and device independence. The QIO system macro format for the logical write function is similar for both the disk and the line printer. Therefore, a user task that dumps large streams of text to a disk via a logical write function would not be affected by the reassignment of its LUN to the line printer if some condition makes this transfer of devices necessary. The parameters stadd and size are the same for the disk and line printer; however, the parameter, comp=0, implies no carriage control

on the line printer (above that already imbedded in the text). The parameters, blkh and blkl are ignored by the line printer handler task.

1.7.6 Status Returns

The symbolic status return codes are used to determine the success or failure of a QIO system macro. The symbolic codes are compared with the value returned in the low order byte of the I/O status block. Status return codes have a two letter prefix of either IE or IS, a period and a three letter suffix. For example, IE.DNR is the symbolic code for the status return that means Device Not Ready. Each device handler chapter contains a list of the symbolic status return codes for the handler. Appendix A contains a complete list of the symbolic codes and their definitions.

1.8 Devices Supported

The chapters that follow in this manual explain in detail the I/O support provided for the standard IAS devices listed below:

- Terminals
- AFC-11 and AD01 analog/digital converters
- Disk
- UDC-11
- DECtape
- DECtape II
- Magnetic tape
- LPS-11
- Card Reader
- Line Printer
- MO pseudo device
- Paper Tape Reader/Punch
- Cassette Tape
- Null Device

1.8.1 Characteristics Words

Each device unit in IAS has four characteristics words that are set or implied at system generation by the DEV directive. These words are stored in the system's Physical Unit Directory (PUD) with offsets U.C1, U.C2, U.C3 and U.C4 from the address of the PUD entry for the particular unit. For task access to the PUD, PUD layout and the layout of words 1 and 4 (offsets U.C1 and U.C4) see the *IAS Guide to Writing a Device Handler Task*, Chapter 2 and Appendix B.

The layout of words 2 and 3 (offsets U.C2 and U.C3) depends on the nature of the device, for example whether the device is random-access or not. Words 2 and 3 are described in the relevant chapters of this manual for devices for which they are defined. For devices for which words 2 and 3 are not described these words are reserved for use by the IAS system.

2 Terminal Handlers

2.1 Terminal Support

Terminal support is provided by the following handlers:

1 The single-terminal handler (TT01) supports one Teletype®-compatible terminal on a DL-11 line with very limited features. (It has no typeahead, no Ctrl/O, no Ctrl/R, no XON/XOFF). It should be used only on minimum configurations where space is at a premium.

The single-terminal handler TT01 is NOT supported in multiuser and timesharing systems.

2 The multiple-terminal handler (TT) supports many terminals on all types of interface and has a large number of additional features.

TT01 is described in Section 2.12.

Where more than one terminal is required TT must be used (See sections 2.2 through 2.10).

The terminal handler is installed with TT... as the task name.

2.1.1 Interface Support

The following standard Communication Line Interfaces are supported:

- KL11 (at 300 baud or less)
- DL11-A,-B,-C,-D,-E
- DJ11
- DH11
- DH11/DM11-BB
- DC11 (at 300 baud or less)
- DZ11, DZQ11, DZV11
- DHV-11, DHV11, DHQ11, DHF11

2.2 Character Input From A Terminal

2.2.1 Special Characters

Most characters are passed directly to the program performing input. Control characters and certain others are used for special purposes and are described below (see Section "Ctrl/B" to Section"Other Special Characters" together with Section "Lower Case Characters").

To input a control character, for example Ctrl/C, press Ctrl and while this key is still down press the key indicated, in this case C.

® Registered Trademark of the Teletype Corporation.

Ctrl/B (Start Paper Tape Input)

On a terminal set with low-speed paper tape reader support, Ctrl/B signals to the computer to start reading the tape.

Ctrl/C

The general function of Ctrl/C is to alert the operating system.

The effect of Ctrl/C depends upon the CLI (Command Language Interpreter) allocated to the terminal, and upon the application tasks running at that terminal.

In a real-time or multi-user system, if MCR is allocated to the terminal, MCR will prompt for command input. If DCL (PDS) is allocated to the terminal, it will be activated or, if it is already active, it will prompt for further input.

In a timesharing system, on an inactive (logged-out) terminal, Ctrl/C activates the Command Language Interpreter (CLI) allocated to the terminal, for example, PDS. On an active terminal the effect is CLI-dependent. Typically, any currently active task is suspended and the CLI will prompt for command input.

In either case, a task running at the terminal may claim Ctrl/C using either the attach-with-ASTS QIO (see Section "IO.ATA"), in real-time or multi-user systems, or the facilities of the Timesharing Control Services (TCS), in timesharing systems. The *IAS Guide to Writing Command Language Interpreters* describes the TCS facilities.

Ctrl/C can affect type ahead in one of two ways:

1 There is no effect. Type-ahead (see Section 2.2.2) remains intact and any read currently under way is unaffected.

2 All type-ahead is flushed. If there is a read under way all characters typed so far are discarded. The read is terminated with a status of IS.CC. If output has been suspended by Ctrl/S it is resumed as though Ctrl/Q had been typed.

Method (1) is normal for real-time and multi-user systems and method (2) for timesharing systems. However, this may be changed for each terminal either at system generation, or dynamically using the appropriate command (see the *IAS PDS User's Guide* or the *IAS MCR User's Guide*.

Ctrl/K, Ctrl/L ,Ctrl/I

These characters may be used in place of vertical tab, form feed and horizontal tab respectively, on terminals which do not have the corresponding keys.

Ctrl/O

While terminal output is in progress, typing Ctrl/O suppresses further output from the same task until one of the following occurs:

1 Another Ctrl/O is typed. Alternate Ctrl/O have the effect of suppressing and enabling output.

2 Another task performs a write to the terminal.

3 A successful attach or detach QIO is performed.

4 A write and cancel Ctrl/O (IO.CCO) is performed.

Ctrl/Q (XON)

To be used to resume output after it has been suspended by Ctrl/S (see Section "Ctrl/S").

Ctrl/R

The effect of Ctrl/R depends on whether or not a read is currently being processed.

If a read is being processed, a "clean copy" is printed of all that has been typed on the line so far with no trace of erased characters. This is particularly useful on hardcopy terminals after many erasures have been made. On timesharing systems, the retyped line is preceded by the prompt, if any.

If there is no read under way, Ctrl/R may be used to check the current line of type-ahead. If "immediate-processing" type ahead is in use (See Section 2.2.2, mode 3), the line currently being typed will be printed.

There is no limit to the number of times Ctrl/R may be typed for a single line.

Ctrl/S (XOFF)

Typing Ctrl/S at any time will temporarily suspend output from the terminal. Unlike Ctrl/O, Ctrl/S does not result in the loss of any output. Output is resumed by typing Ctrl/Q (or Ctrl/C, see Section "Ctrl/Z").

Ctrl/T (Terminate Paper Tape Input)

See Section 2.10.

Ctrl/U

Typing Ctrl/U will cancel all characters typed on the line so far while a read is in progress. On timesharing systems, the prompt (if any) is repeated. The line may then be started again.

Ctrl/V

Typing Ctrl/V will flush all type-ahead. There is no other effect. If there is a read in progress Ctrl/V has no effect.

Ctrl/X

On only real-time and multi-user systems typing Ctrl/X will invoke a terminal-specific task called TTYNxx, where xx is the terminal number. If a task of this name is not installed there is no effect. Any read or type-ahead remains unaffected.

Ctrl/Z

This character terminates the current input line with a status of IE.EOF (end-of-file), and echoes as "^Z". Any characters already typed in the line are passed to the program doing the read.

Ctrl/?

On a VT61 set in escape sequence mode, Ctrl/? has the effect of ALTMODE (see Section "Altmode").

Carriage Return

This character terminates the current line of input, with a status of IS.CR.

ALTMODE (Escape)

This character also terminates the current line of input, but with a status of IS.ESC. Some older Teletype* devices produce a non-standard character when the "ALTMODE" key is pressed. For this to be recognized the terminal must have been set up appropriately (at system generation or by the TER (MCR) or SET TERMINAL (DCL) commands).

Normally this character is echoed as "$" followed by carriage-return. However, when the handler is built as part of the system generation process it is possible to specify that no echo at all be produced.

This gives compatibility with earlier versions of IAS and RSX-11D, and RSX-11M.

A terminal may be set up as an "escape sequence" terminal. In this case the effect of escape is as described in Section 2.8. See also Section '[Ctrl/?]'.

Rubout

This character erases the last character typed on the current input line. Each time it is typed, a further character is erased, until all characters on the line have been removed. On a hard-copy terminal, the characters erased are enclosed between backslashes (a horizontal tab is printed as backslashes two spaces). On a VDU, the character is physically removed, and the cursor is left where it was before the character was typed.

Other Special Characters

All remaining special characters can be divided into two groups:

1 Characters that are echoed (as themselves) and passed to the program that requested the read. This group consists of [Ctrl/G] (bell), form feed, vertical tab and line feed.

2 Characters that are ignored. This group includes all other characters whose ASCII code is less than 40 (octal).

Lower Case Characters

Lower case characters in this context are those that are in the 96-character ASCII set, but not in the 64-character set. This includes not only the lower case letters but also "`", "{", "}", " | ", and "~".

If the terminal is set as "NOLOWERCASEKEYBOARD", all lower case characters are converted to their upper case counterparts (the non-alphabetic characters become "@", "^", "[", "]", "\", and "^",) as they are read.

If the terminal is set as "LOWERCASEKEYBOARD", and "NOLOWERCASEINPUT", characters are normally converted to upper case as they are read. If type-ahead [Ctrl/R] is used, the characters will be printed in lower case even though they will be seen as upper case by the reading program. If a read with no case conversion (TF.RNC) is used, lower case characters will be passed.

If the terminal is set as "LOWERCASEINPUT", lower case characters will be echoed and passed on as such to tasks doing input, even if they do not specify TF.RNC.

2.2.2 Type-ahead

The name "type-ahead" refers to characters that are typed while there is no read in progress from the terminal. The terminal handler can be set, for each terminal, to process type-ahead in one of three ways:

1 Ignore type-ahead. Any typed-ahead characters result in a "BELL" code being sent to the terminal, but are otherwise ignored. This mode of operation is the most suitable for inexperienced users who may find other modes confusing.

2 Store type-ahead exactly as typed and process it only when it is obtained by a read request ("deferred processing" mode). This is the simplest mode to understand. It means, for example, that if rubouts or [Ctrl/U] are typed ahead, they are echoed exactly as though they had been typed in response to the prompt. Type-ahead [Ctrl/R] is ineffective since the [Ctrl/R] character

is not processed until the read is under way. A maximum of 80 (decimal) characters may be typed-ahead.

3 Perform some processing as characters are typed, but echo only when they are read by a task ("immediate processing, deferred echo" mode). Characters such as Ctrl/U and rubout are processed immediately, so that when the line is echoed a "clean copy" is seen. This mode produces the cleanest, most legible log of console operation and makes it clear which type-ahead has been read and which is still in the handler's buffer. If a user believes that a typing mistake may have been made, the type-ahead Ctrl/R facility can be used to inspect the current line.

   Problems may arise in connection with programs such as ODT which use read-pass-all (Section "TF.RAL") to perform their own non-standard character processing. When the handler detects a read-pass-all request, it temporarily switches to "deferred processing" mode. If characters are typed ahead before the program runs, then Ctrl/U and rubout, in particular, will be processed in "immediate processing" mode. This situation is unlikely to arise a great deal in practice, but if it does, "deferred processing" mode may be more appropriate.

The normal default mode is (3), "immediate processing", but this may be changed during system generation.

Regardless of read-ahead type, the characters Ctrl/C, Ctrl/O, Ctrl/S (XOFF) and Ctrl/Q (XON) are effective as soon as they are typed. Section 2.11.7 describes additional ways of processing these characters.

2.3 Character Output to a Terminal

Most characters are simply copied directly from the user's buffer to the output device or from the user's input in the case of echo. Some characters are treated specially by the handler as described below. For a "write pass all" request (Section "TF.WAL") all characters are passed with no interpretation.

2.3.1 Escape (ALTMODE)

Escape, character code 33 (octal), is passed unchanged to the terminal.

2.3.2 Form Feed

This character is normally replaced by six line feeds. However, a terminal may be set to simulate form feed so that the effect is as it would be on a lineprinter. It is also possible to specify that the device has hardware form feed, in which case no interpretation is provided.

2.3.3 Horizontal Tab

For terminals which do not have hardware horizontal tab, this character is simulated to provide tab stops every eight character positions across the page. Rubout after horizontal tab causes the handler to remove the necessary number of spaces on a scope.

If a terminal is specified as having hardware horizontal tabs, the handler assumes that tab stops are at every eight spaces, unless the terminal is also specified as having non-standard tabs.

The VT05 and VT5x show rubout over tab correctly on the screen except when the tab is in one of the last eight character positions in the line. On VDUs with non-standard hardware tabs, rub out removes one space. If true cursor positioning is essential in such cases the terminal should be set as having no hardware tabs, so that the tabs are simulated by software.

2.3.4 Line Feed

This character normally has an implicit Carriage Return associated with it. To advance the paper without returning the print position to the left hand margin the character must be output in write-pass-all mode (see Section "TF.WAL").

2.3.5 Lower Case

As described in Section "Carriage Return", "lower case" includes certain non-alphabetic characters as well. If the terminal is set as "LOWERCASEOUTPUT", lower case characters are passed intact but otherwise they are translated to their upper-case counterparts.

2.3.6 Vertical Tab

This character is normally replaced by four line feeds. If form feed simulation is in effect the number of linefeeds necessary to reach the next vertical tab stop is output. Vertical tab stops are assumed to be every six lines, except at the bottom of the page.

2.4 Function Codes

2.4.1 Read

The basic read function is IO.RLB (Read Logical Block). The general format of a read request is:

```
QIO$   fc,lun,ef,pri,iosb,ast,<stadd,size,tmo>
```

where:

- "stadd" - is the start address of the user buffer and may be odd or even.
- "size" - is the buffer size in bytes, which must be non-zero and less than 8128 (decimal).
- "tmo" - is the timeout for the read, in units of approximately ten seconds (with TF.TMO only).

The IO.RLB function may be modified by "or"ing it with one or more of the following sub-function codes using the logical "or" operator ("!").

TF.RAL (Read pass all)

All characters, including control characters, null and rubout, will be placed in the user buffer. Since no characters are recognised as terminators the read will only complete when the buffer is full or if an error condition arises. For most purposes therefore the buffer size should be just one byte. If all eight bits of the characters are to be passed in the buffer, the terminal characteristic "P8B" should be set, otherwise the parity bit will be removed. The characters Ctrl/C, Ctrl/O, Ctrl/Q, Ctrl/S and Ctrl/X will not however be passed to the requesting task, but will have their usual effect. This may be altered by setting the terminal as "BINARY", see Section 2.11.6.

TF.RNE (Read with No Echo)

No characters are echoed, not even carriage return. Rubout and Ctrl/U have their usual effect but produce no output. Ctrl/R has no effect. This function may be used when a program wishes to perform its own echoing (normally in conjunction with read-pass-all, above), or where the information being input must be kept secure, for example, passwords.

TF.RNC (Read with No Case Conversion)

If the terminal is set to LOWERCASEKEYBOARD and NOLOWERCASEINPUT, lower-case characters will normally be converted to upper case. This function code overrides the conversion so that lower-case characters may be read.

TF.TMO (Read with Timeout)

"TMO" is the time, in units of approximately ten seconds (plus or minus 0.5 seconds), which may be allowed to elapse between successive characters being typed. If this time is exceeded the read is terminated with a status of IS.TMO. All characters previously typed are passed in the buffer. Only the low byte of "tmo" is significant. The high byte is reserved for future use and must be zero.

If a task issues a read request with "tmo" set to zero, the request will always be completed immediately. If one or more completed records of type-ahead are available, the first will be read in the usual way. Otherwise, all available characters will be read, with a status of IS.TMO. This facility may be used, for example, to ascertain whether input is available without waiting if it is not, or, by using it repeatedly until no more characters are returned, to flush and ignore type-ahead.

Features of Read Function Codes

The function code IO.RVB (read virtual block) may be used instead of IO.RLB.

With IO.RLB any of the modifiers may be combined, using the logical "or" operator ("!"), except that if TF.RAL is specified the only other values permitted are TF.RNE and TF.TMO.

A read request is terminated by one of the following events:

1 The user buffer is filled. The status in the first word of the I/O status block is IS.SUC.

2 One of the terminator characters carriage-return or altmode is typed. The status values are IS.CR and IS.ESC, respectively.

3 An escape sequence is typed. See Section 2.7 for a discussion of escape sequences.

4 The timeout limit is reached between characters (TF.TMO only). The status return is IS.TMO.

5 One of the error conditions described in Section "Read Error Conditions" is detected.

The second word of the I/O status block always contains the number of bytes transferred into the user buffer, even after an error condition.

If the number of characters input before a terminator is exactly equal to the buffer size, the request will be terminated with a status of IS.SUC. A subsequent read will receive a status corresponding to the terminator, with a character count of zero.

Read Error Conditions

The following error conditions may arise for a read request:

- IE.ABO - The handler was unloaded while the request was pending or the request was aborted by IO.KIL.

- IE.BCC - A framing error occurred. IE.BCC will be returned if:
  1 the line becomes disconnected from the terminal
  2 the "break" key is depressed
  3 the line connecting the terminal to the computer is faulty
  4 the terminal speed is set incorrectly
- IE.DAO - A data overrun error occurred, that is the characters were received from the terminal more quickly than the computer could handle them. IE.DAO normally indicates that the processor is severely overloaded.
- IE.DNR - The request was made to a dialup line which is not connected.
- IE.EOF - This is not strictly an error. It means that Ctrl/Z has been typed at the terminal.
- IE.FHE - An internal buffering error has occurred in the handler. This may occur if the handler has been built with too small a node pool.
- IE.OFL - The terminal was specified in System Generation Phase 1 and has a PUD entry, but its interface is not physically present in this configuration.
- IE.SPC - The specified buffer (or prompt string) is wholly or partially outside the user's address space or is longer than 8128 (decimal) bytes.
- IE.VER - A character had incorrect parity. The offending character is lost.

Read with Prompt (IO.RPR)

The basic read-with-prompt function code is IO.RPR. The general format is:

```
QIO$   fc,lun,ef,pri,iosb,ast,<stadd,size,tmo,pradd,prsize>
```

where:

- stadd - is the start address of the user buffer and may be odd or even.
- size - is the buffer size in bytes, which must be non-zero and less than 8128.
- tmo - is the timeout for the read, in units of approximately ten seconds.
- pradd - is the start address of a user buffer containing a prompt string. It may be odd or even.
- prsize - is the length, in bytes, of the prompt string. It must be non-zero and less than 8128.

All the subfunctions and error returns noted in Sections "TF.RAL" through "Read Error Conditions" apply equally to IO.RPR as to IO.RLB. For example, to issue a read-with-prompt and timeout, you specify the TF.TMO subfunction (See Section "TF.TMO").

The function IO.RPR performs a read immediately preceded by a prompt. It has the following advantages over performing a separate write before the read:

1 Only a single QIO is needed, reducing both the complexity of the program and the system overhead.

2 There is no possibility of two tasks simultaneously prompting and trying to read, leaving the user unsure which task is receiving the input, since the prompt and the read are not separable.

3 If the user types Ctrl/U or Ctrl/R the prompt will be repeated. It will also be repeated if the read is interrupted by a write (see Section "TF.WBT").

There is no implicit carriage control in the prompt. In particular, if the prompt is to appear on a new line, it must include the characters carriage return and line feed.

IO.RPR is NOT equivalent to a write (IO.WLB) followed by a read (IO.RLB).

1 The handler assumes that the prompt string will be fairly short and does not take the same precautions against running out of internal buffer space as it does for a write.

2 The prompt will be repeated under the circumstances noted above.

3 There is no provision for a vertical format character in the prompt.

The following example will perform a read-with-prompt (INPUT:). If no input is received in ten minutes, the read will be terminated.

```
PROMPT: .ASCII  <  15  >
 <  12  > /INPUT: /   ;PROMPT
PRSIZE=.-PROMPT
     .EVEN
TMO=10.*6                ;TIMEOUT = 10 MINUTES
IOSB:   .BLKW  2
BUF:    .BLKW  100.
BSIZ=.-BUF
START:
   QIOW$S #IO.RPR!TF.TMO,#5,#1,,#IOSB,,<#BUF,#BSIZ,#TMO,#PROMPT,#PRSIZE>
```

2.4.2 c2_Write

The basic write function is IO.WLB (write logical block). The format of a write request is:

```
QIO$  fc,lun,ef,pri,iosb,ast,<stadd,size,vfc>
```

- "stadd" - is the start address of the buffer (which may be odd)
- "size" - is the buffer size in bytes, which must be less than 8128 (decimal)
- "vfc" - is the vertical format character, which specifies the paper spacing action to be taken for this write. The legal values are described in Table 2–1. It may also be an escape sequence identifier; see Section 2.7.2.

The IO.WLB function may be modified by "or"ing it with one or more of the sub-function codes that follow in Sections "TF.WAL" to "TF.SYN".

Table 2–1 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.
060	0 (zero)	DOUBLE SPACE - Output two line feeds, print the contents of the buffer, and output a carriage return. The buffer contents are printed two lines below the previously printed line.

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

Table 2–1 (Cont.) Vertical Format Control Characters

Octal Value	Character	Meaning
061	1 (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. See Section 2.3.2.
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 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. However it is recommended that the IO.RPR function be used to perform a read where a prompt is required.
000	null	INTERNAL VERTICAL FORMAT - The buffer contents are printed without addition of vertical control characters. In this mode, more than one line of guaranteed contiguous output can be printed per I/O request queued.

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

TF.WAL (Write Pass All)

All characters in the buffer are passed directly to the terminal. It is the user's responsibility to provide filler characters where required, to simulate tab functions and so on. This function must be used to output cursor control sequences where the characters do not have their usual significance. After a write-pass-all request, the handler loses track of the horizontal and vertical position. Also, if a read follows and is to start on a newline, a line feed must be output explicitly. If the terminal is set for parity generation, bit 8 of the character will be replaced by the parity bit for bits 0-7. Otherwise, all eight bits are passed to the terminal.

TF.CCO (Cancel Ctrl/O)

If Ctrl/O is in effect it is cleared before the write is commenced. This ensures that the output will appear on the terminal.

TF.WMS (Write Message)

Normally a task running under [1,1] can perform a write to another terminal even if it is set in "NOMESSAGES" mode (see Section "Features of Write Function Codes". By using this variant of the write function the request is treated as though it came from a task not running under [1,1].

TF.SYN (Synchronous Mode)

Normally, "I/O done" is performed for a write request as soon as it is queued (see below). However, for some applications it is important to know when the final character has been output, for example, so that the time at which a following read begins can be known accurately. If the subfunction code TF.SYN is set, I/O completion will not occur until the last character has been output. The IO.KIL function issued before completion has occurred will abort the write immediately.

TF.WBT (Write Break Through)

The subfunction TF.WBT causes a write to be processed even if a read is under way and some characters have been typed. The action is:

1 ^R is printed at the end of the characters typed so far.

2 The write is performed.

3 The previously typed characters are repeated (unless the read specified no echo), preceded by the prompt, if any.

If output at the terminal has been suspended by Ctrl/S, TF.WBT will not cause it to be resumed. Only Ctrl/Q typed at the terminal has this effect.

Features of Write Function Codes

The function code IO.WVB (Write virtual block) may be used instead of IO.WLB.

More than one modifier may be combined on a single request, using the logical "or" operator.

The handler normally buffers the characters to be written internally, and is able to complete the request as soon as it is issued. This means that the event flag (if any) specified in the QIO will be set as soon as the directive is issued, and the AST (if any) will be obeyed as the next instruction after the directive. A user task must not depend on a delay between issuing the request and being notified of its completion. Exceptionally, if the handler is heavily loaded and buffer space is limited, there may be a delay. On timesharing systems only, checks are made before a write request is performed. If the terminal is set in "NOMESSAGES" mode, only write requests from a task running on that terminal or from tasks running under [1,1] will be accepted. Other write requests will be rejected with a status of IE.PRI.

If a read request is outstanding on a terminal when a write is issued, the read may be suspended while the write is performed. This will only occur if nothing has been typed on the keyboard for this read. If the read was a read-with-prompt (IO.RPR), the prompt will be repeated when the write is complete.

On the operator's terminal, that is, the terminal to which Console Output (CO) is redirected, a write will break through a read at any time. This will also happen if the write specified the subfunction TF.WBT (Write Break Through) (see Section "TF.WBT").

When a write request is completed, the first word of the I/O status block contains the success or failure code (see Section "Write Error Conditions"), and the second word contains the number of bytes transferred.

Write Error Conditions

The following error codes may be returned for a write request:

- IE.ABO, IE.DNR, IE.OFL, IE.SPC - These have the same meaning as for a read (see Section "Read Error Conditions")
- IE.PRI - The terminal was set in "NOMESSAGES" mode.

2.4.3 Set/Get Terminal Characteristics

This group of functions allows a user task to discover the characteristics of a terminal or to change them dynamically. There are a number of different functions which are discussed separately below. The general format is:

```
QIO$   fc,lun,efn,pri,iosb,ast,<p1,p2,p3,p4,p5,p6>
```

Features of Set/Get Characteristics

To avoid the proliferation of highly specific function codes, the names of these functions are of the form SF.xxx. These symbols and all others specific to this section are defined in the module TTSYM which is automatically extracted from SYSLIB, if required. These symbols may also be defined locally using the macro TTSYM$ which may be defined using the .MCALL directive.

When a set characteristics (as distinct from get characteristics) is performed, the current state of the terminal characteristics will be saved by the handler, unless this has already been done. This means that a user can change the terminal characteristics but they can subsequently be restored, for example on logout, by the function SF.RDF (Section "SF.RDF").

It may be necessary to make a permanent change in characteristics, for example, if a terminal is replaced with a different model. This is done by "OR"ing the value SF.DEF and the function code. There is an implicit restore defaults (SF.RDF) before the change is made, in this case.

Only a task running on a terminal, or one running under a UIC of a [1,1], is allowed to set the characteristics of that terminal. Only a task running under [1,1] may change the default characteristics.

The set of characteristics is listed in Table 2–2. Each has a name of the form TC.xxx by which it is always referenced. Many characteristics are binary valued, that is their value must be 0 or 1. Others may take a value in a range. The terminal type (TC.TTP) and line speed (TC.RSP and TC.XSP) must have values selected from the symbolic names in Table 2–3 and Table 2–4 respectively.

Some characteristics can be read but cannot be changed dynamically. These are indicated as "fixed" in Table 2–2.

A set or get characteristics function is always performed immediately, even if the terminal has a read or a write in progress. If a set characteristics request immediately follows a write request, the write should normally include the function qualifier TF.SYN ("TF.SYN") to ensure that all characters have been output before any change is made.

Table 2–2 Characteristics and their Names

Name	Description	Max. Val. (Note 1)	Fixed?	Depends on (Note 2)	Note
TC.ABD	Autobaud detection	B	N	$$ABD	
TC.ACR	An automatic carriage return/line feed is to be supplied when a character to be printed would go beyond the end of the physical line.	B	N		
TC.EDT	Terminal performs edit functions.	B	N		
TC.ALT	Terminal requires recognition of the alternative altmode characters 175 and 176 (octal). B	N			
TC.ANI	ANSI CRT terminal	B	N		
TC.ANS	Terminal is to operate in ANSI escape sequence mode. If ANS is not set then VT52 escape sequence mode is used.	B	N		15
TC.AVO	VT-100-family terminal display	B	N		
TC.BIN	If terminal is set in deferred processing mode, all characters are passed for a read-pass-all.	B	N		14
TC.BLK	The terminal is a VT61 and is to operate in block mode.	B	N	B$$LCK	11
TC.BSP	Terminal recognizes and is able to interpret the "back space" character	B	N	B$$SP	8
TC.CEQ	If ESQ is specified and CEQ is set, escape sequences are to be input in compatible mode (see Section 2.7.1).	B	N	E$$SEQ	
TC.CCF	Ctrl/C flushes all input (see Section "Ctrl/B")	B	N		
TC.CSQ	If terminal is set in deferred processing mode, all characters except Ctrl/S and Ctrl/Q are passed for a read-pass-all.	B	N		14
TC.CTC	If terminal is set in deferred processing mode, all characters except Ctrl/C, Ctrl/S and Ctrl/Q are passed for a read-pass-all.	B	N		14
TC.DEC	Digital CRT terminal	B	N		
TC.DLU	Line is a dialup line	B	Y	D$$IAL	
TC.EDT	Terminal performs editting functions.	B	N		
TC.EPA	If PAR is specified and EPA is set, parity should be even, otherwise it should be odd.	B	N		
TC.ESQ	Terminal requires escape sequence support (see Section 2.7).	B	N	E$$SEQ	

Table 2–2 (Cont.) Characteristics and their Names

Name	Description	Max. Val. (Note 1)	Fixed?	Depends on (Note 2)	Note
TC.FDX	Terminal is to operate in full duplex mode (see Section 2.11.5).	B	N		
TC.FRM	The terminal is a VT61 and is to operate in forms mode.	B	N	B$$LCK	11
TC.HFF	Terminal recognizes and is able to interpret form feed and vertical tab.	B	N		
TC.HFL	Horizontal fill requirement.	7	N		7
TC.HHT	Terminal recognizes and is able to interpret horizontal tab, and does not require software simulation.	B	N		8
TC.HLD	The terminal is a VT5x or a VT61 and is to operate in hold screen mode.	B	N	E$$SEQ or H$$OLD	11
TC.IMG	Messages sent from a task running at another terminal are to be rejected.	B	N		12
TC.ISL	Subline on interface.	15	Y		
TC.LCP	Device has local copy, i.e. echoes every character itself as typed.	B	N		
TC.LPP	Length of page in lines.	255	N	S$$FF	
TC.LVF	Terminal is an LA36 with vertical format option. Form feed and vertical tab will be followed by 66 null fillers.	B	N		
TC.NEC	Echo suppressed	B	N		
TC.NKB	Terminal is not enabled for input; read requests will be rejected.	B	N		
TC.NL	Terminal generates "newline" instead of "carriage return".	B	N	N$$L	
TC.NPR	Terminal is not enabled for output; write requests will be rejected.	B	N		
TC.NST	Terminal has the "HHT" attribute but does not provide the standard interpretation (tab stops every 8 spaces).	B	N		
TC.PAR	Line requires parity checking on input and parity generation on output.	B	N		10
TC.P8B	All 8 bits of the character will be passed for a read- pass-all request. The terminal should also be set to "deferred processing" read ahead mode	B	N		
TC.RAT	Read-ahead type.	2	N		5

Table 2–2 (Cont.) Characteristics and their Names

Name	Description	Max. Val. (Note 1)	Fixed?	Depends on (Note 2)	Note
TC.RGS	Terminal supports REGIS instructions	B	N		
TC.REM	Line is remote.	B	Y	D$$IAL	
TC.RSP	Receiver speed (keyboard speed of terminal).		N		4
TC.SCP	Scope (rubout can physically erase characters).	B	N		
TC.SFF	Full simulation of form feed and vertical tab should be provided instead of providing just a "token".	B	N	S$$FF	
TC.SMO	Enable lower-case output.	B	N		
TC.SMP	Force lower-case input.	B	N		9
TC.SMR	Enable lower-case input.	B	N		9
TC.TAP	The terminal has a low speed paper tape reader. See Section 2.9.	B	N	T$$APE	
TC.STB	Extra stop bit required.	B	N		
TC.TTP	Terminal type.		N		6
TC.UC0 through TC.UC9	User-definable characteristics	B	N		13
TC.VFL	Vertical fill (6 nulls) required after line feed, form feed or vertical tab.	B	N		
TC.WID	Width of page or screen in characters.	255	N		3
TC.XSP	Transmitter speed (receiver speed of terminal).		N		4
TC.8BC	Pass eight bits on input, even if not binary input mode.	B	N		

Notes referred to by number in columns 3, 5 and 6 of Table 2–2.

1 The minimum value of every characteristic is 0. "B" indicates that the characteristic is binary with a value of 0 or 1.

2 If the assembly parameter specified in column 5 is zero, this characteristic is not available and any attempt to reference it will return an error of SE.NIH.

3 The value specified or returned is actually one greater than the width in characters of the device, for example, 81 for an LA30.

4 The value of a line speed characteristic must be one of the values in Table 2–4.

5 The value specified or returned corresponds to one of the three read-ahead types described in Section 2.2.2, that is:

- 0 - ignore type-ahead
- 1 - deferred processing

- 2 - immediate processing

6 The value specified or returned corresponds to the values in Table 2–3. Note that simply setting this characteristic, by SF.SSC or SF.SMC, will not perform the implicit changes described in Section "SF.STT, SF.STS".

7 If this value is in the range 1-6, it specifies the number of null fillers to be supplied after a carriage return. If it is zero, no fill is supplied. If it is 7, the fill is suitable for an LA30S, provided the assembly parameter L$$30S is non-zero. See also note 8.

8 If TC.HFL, the horizontal fill count, is non-zero, a single null fill will be supplied after the character.

9 TC.SMR corresponds to "LOWERCASEKEYBOARD" and TC.SMP to "LOWERCASEINPUT". See Section "Lower Case Characters".

10 This characteristic is only available on interfaces which perform hardware parity checking and generation.

11 If a terminal which is not of one of the stated types has this characteristic set to 1, no error will be returned but there will be no effect.

12 This corresponds to NOMESSAGES mode. See Section "Features of Write Function Codes".

13 These characteristics are available for user modifications to the terminal handler. In the Digital-supplied version, they are not given specific meanings.

14 See Section 2.11.7 for description of effects of reading control characters from terminals with these characteristics.

15 The terminal must also be set to have escape sequence support (TC.ESQ set) and the terminal handler parameter E$$SEQ must be non-zero. See also Section 2.8.

Table 2–3 Valid Terminal Types

Name	Terminal Type	Implicit Characteristics (Note 1)	Width	Length	Speed (baud)
T.AS33	ASR33	TC.HFL=1, TC.STB	72	66	110
T.KS33	KSR33	TC.HFL=1, TC.STB	72	66	110
T.AS35	ASR35	TC.HFL=1, TC.STB	72	66	110
T.L30S	LA30S	TC.HFL=7	80	66	300
T.L30P	LA30P	none	80	66	300
T.LA36	LA36	TC.ACR, TC.BSP, TC.LVF, TC.SMO, TC.SMR	132	66	300
T.LA34	LA34	TC.HFL, TC.BSP, TC.SMO, TC.SMR, TC.ACR	132	66	300
T.LA38	LA38	TC.HFL, TC.BSP, TC.SMO, TC.SMR, TC.ACR	132	66	300
T.LA100	TA100	TC.HFL, TC.HFF, TC.BSP, TC.SMO TC.SMR, TC.ACR	132	66	1200
T.VT05	VT05	TC.ACR, TC.BSP, TC.HHT, TC.SCP, TC.VFL	72	20	2400
T.VT50	VT50	TC.ACR, TC.BSP, TC.HHT, TC.SCP	80	12	9600

Table 2–3 (Cont.) Valid Terminal Types

Name	Terminal Type	Implicit Characteristics (Note 1)	Width	Length	Speed (baud)
T.VT52	VT52	TC.ACR, TC.BSP, TC.HHT, TC.SCP, TC.SMO, TC.SMR	80	24	9600
T.VT55	VT55	TC.ACR, TC.BSP, TC.HHT, TC.SCP, TC.SMO, TC.SMR	80	24	9600
T.VT61	VT61	TC.ACR ,TC.BSP ,TC.HHT ,TC.SCP, TC.SMO, TC.SMR	80	24	9600
T.L180	LA180S	TC.HFF, TC.HFL=6, TC.SMO	132	66	2400
T.L120	LA120	TC.ACR, TC.BSP, TC.LVF, TC.SMO, TC.SMR	132	66	1200
T.V100	VT100	TC.ACR, TC.BSP, TC.HHT, TC.SCP, TC.SMO, TC.SMR, TC.ANI, TC.DEC	80	24	9600
T.V101	VT101	TC.SCP, TC.HHT, TC.BSP, TC.ACR, TC.SMR, TC.SMO, TC.ANI, TC.DEC	80	24	9600
T.V102	VT102	TC.SCP, TC.HHT, TC.BSP, TC.ACR, TC.SMR, TC.SMO, TC.ANI, TC.DEC, TC.EDT	80	24	9600
T.V105	VT105	TC.SCP, TC.HHT, TC.BSP, TC.ACR, TC.SMR, TC.SMO, TC.ANI, TC.DEC, TC.RGS	80	24	9600
T.V125	VT125	TC.SCP, TC.HHT, TC.BSP, TC.ACR, TC.SMR, TC.SMO, TC.ANI, TC.DEC	80	24	9600
T.V131	VT131	TC.SCP, TC.HHT, TC.BSP, TC.ACR, TC.SMR, TC.SMO, TC.ANI, TC.DEC	80	24	9600
T.V132	VT132	TC.SCP, TC.HHT, TC.BSP, TC.ACR, TC.SMR, TC.SMO, TC.ANI, TC.DEC, TC.BLK	80	24	9600
T.V2XX	VT2XX	TC.SCP, TC.HHT, TC.BSP, TC.ACR, TC.SMR, TC.SMO, TC.ANI, TC.DEC, TC.AVO, TC.EDT	80	24	9600
T.USR0 through T.USR4		(Note 2)			

Notes on Table 2–3:

1 Unless otherwise stated, the following characteristics are set to 0 (or "NO") whenever an SF.STT or SF.STS request is made:

2 TC.BSP, TC.ESQ, TC.HFF, TC.HFL, TC.HHT, TC.LCP, TC.LVF, TC.NL, TC.NST, TC.SCP, TC.SFF, TC.SMO, TC.SMR, TC.STB, TC.VFL.

3 These names correspond to user-defined terminal types which may be included when the terminal handler is built (see the *IAS Installation and System Generation Guide*).

The value for the TC.RSP and TC.XSP characteristics must be one of the following symbols.

Table 2–4 Valid Terminal Speeds

Name	Speed (baud)
S.0	0 (line disabled)
S.50	50
S.75	75
S.100	100
S.110	110
S.134	134.5
S.150	150
S.200	200
S.300	300
S.600	600
S.1200	1200
S.2000	2000
S.2400	2400
S.3600	3600
S.4800	4800
S.7200	7200
S.9600	9600
S.EXTA	DH11 External speed rate A
S.EXTB	DH11 External speed rate B

SF.SSC (Set Single Characteristic)

This sets a single characteristic to a specified value. The parameters are:

- p1 - characteristic name
- p2 - value to which it should be set

SF.SMC (Set Multiple Characteristics)

This sets several characteristics in a single operation. It is essential where two or more characteristics must be changed simultaneously. For example it must be used to change the line speed, represented as the two separate characteristics transmit speed and receive speed, on an interface that cannot handle split-speed lines. The parameters are:

- p1 - address of a buffer containing a list of characteristic/value pairs (see below)
- p2 - length in bytes of buffer (must be even, non-zero and less than 8128 (decimal)).

The buffer is a list of byte pairs of the form:

- .BYTE - characteristic name
- .BYTE - new value

SF.STT, SF.STS (Set Terminal Type)

This sets the terminal type (name TC.TTP) and also sets all other characteristics to the appropriate value for that terminal type, for example width, length and fill requirements. SF.STS also sets the line speed to the default value for that terminal type, for example 300 baud for an LA30. SF.STT does not affect the line speed. The parameters are:

- p1 - terminal type (see Table 2–4)

SF.GSC (Get Single Characteristic)

This function is complementary to SF.SSC. It enables a program to sense the value of a single characteristic. The parameters are:

- p1 - name of characteristic to sense

The current value of the specified characteristic is returned in the second word of the I/O status block.

SF.GMC (Get Multiple Characteristics)

This function is complementary to SF.SMC. It enables a program to sense the values of several characteristics simultaneously. The parameters are:

- p1 - address of a buffer in the format described below
- p2 - length, in bytes, of buffer (must be even, non-zero and less than 8128 (decimal).

The buffer format is a list of byte pairs of the form:

- .BYTE - characteristic name
- .BYTE - space to receive value

SF.GAC (Get All Characteristics)

This function dumps all the characteristic settings for a terminal into a specified 16 word buffer. It is intended to be used in conjunction with SF.SAC (below) for programs which modify characteristics and want to reset them on exit even though their value may not be the default. The characteristics are returned in their internal binary format which cannot be interpreted by a user program; the significance of the individual bits varies depending upon the parameters used when the handler is built. The parameters are:

- p1 - address of 16 word buffer to receive characteristics

SF.SAC (Set All Characteristics)

This function takes a buffer filled by SF.GAC (above) and sets all characteristics (except those which cannot be changed) accordingly. The parameter is:

- p1 - address of 16 word buffer

Restore Default Characteristics

This function, which takes no parameters, restores the default values of the characteristics saved when the first "set characteristic" was performed. If no default value has been saved, because no changes have been made, this function has no effect.

Error Conditions of Set/Get Characteristics

Some errors may occur which are common to those reported for other functions. These are:

- IE.IFC - The handler has been built without the specified function (see the *IAS System Generation and Startup Guide*).
- IE.PRI - The requesting task is not running on the same terminal or not running under [1,1].
- IE.OFL - The terminal was specified in system generation phase 1 and has a PUD entry, but its interface is not physically present in this configuration.
- IE.SPC - The specified buffer is wholly or partially outside the user's address space, is zero bytes long or is larger than 8128 (decimal) bytes.

Other errors are specific to this set/get group of functions. These are distinguished by an error code of IE.ABO in the low byte of I/O status block word 1, with an error qualifier in the high byte. These qualifiers are:

- SE.ICN - Illegal characteristic name. A characteristic name was not one of the symbols "TC.xxx" specified in Table 2–2.
- SE.FIX - Attempt to change one of the characteristics specified in Table 2–2 as being fixed.
- SE.BIN - The value specified for a binary characteristic is not 0 or 1.
- SE.VAL - The value specified for a non-binary characteristic is outside the permitted range.
- SE.TER - The terminal type specified in an SF.STT or SF.STS function is not one of the "T.xxxx" symbols specified in Table 2–3.
- SE.SPD - An attempt has been made to set the speed of a line to a value which is not available on the interface to which it is connected.
- SE.SPL - An attempt has been made to set a different value for the receive and transmit speeds on an interface which does not support split-speed lines.
- SE.PAR - An attempt has been made to set a parity-checking type that is not supported by the line interface involved.
- SE.LPR - Some other parameter has been changed which cannot be supported by the interface for a line.
- SE.NSC - A line parameter change (for example, speed) has been specified but the interface does not have settable characteristics.
- SE.UPN - The handler does not have enough internal buffer space to save the default characteristic settings.
- SE.NIH - The specified characteristic does not exist in this handler. The parameter file used when the handler was built indicated that this characteristic was not required.

After one of the "multiple characteristic" functions (SF.SMC and SF.GMC) I/O status block word 2 contains the offset in the user's buffer of the name of the offending characteristic. If the handler is unable to detect which characteristic caused the error, a value of -1 will appear instead.

2.5 Other Functions Affecting Terminals

The functions Attach Terminal, Detach Terminal, Get Terminal Support, Kill Outstanding Requests, and Disconnect Dialup Line are also supported.

IO.ATT and IO.ATA (Attach Terminal with AST Notification)

This is an ATTACH function and specifies asynchronous system traps (ASTs) to process unsolicited TT input. Control passes to the AST. When the task receives an unsolicited character (other than Ctrl/Q, Ctrl/S, Ctrl/X, or Ctrl/O). When specifying a Ctrl/C AST on a timesharing system, the AST will be checked for the required privileges before the AST address is passed to the TCP (Timesharing Control Primatives). Thus, it is not necessary to include the CTC$T call to TCP in your application task. when you detach your terminal, TCP will cancel the Ctrl/C AST.

This QIO is presented in two formats. The first format, Unsolicited Input Line AST, provides only notification of completed, unsolicited lines. The second format, Unsolicited Character AST, passes each character as it received.

The format for Unsolicited Input Line is:

```
QIO$   IO.ATA,lun,ef,pri,iosb,ast,<ccae,uiae>
```

where:

- ccae - is the address of the AST entry point to be entered if Ctrl/C is typed at the terminal. In this case the Ctrl/C will not re-activate the command level program. Therefore, this facility should be used with extreme caution, since if the program loops, it will not be possible to use commands to abort it from the same terminal. For timesharing systems, this parameter requires Ctrl/C privileges.
- uiae - is the address of the AST entry point to be entered if a line of unsolicited input is typed, such that a read request will be completed immediately.

When a Ctrl/C or unsolicited input AST occurs, the task's stack will contain the following values:

- SP+14 - Event flag mask word for flags 1 to 16
- SP+12 - Event flag mask word for flags 17 to 32
- SP+10 - Event flag mask word for flags 33 to 48
- SP+06 - Event flag mask word for flags 49 to 64
- SP+04 - PS of task prior to AST
- SP+02 - PC of task prior to AST
- SP+00 - Task's Directive Status word

No extra parameters are put onto the task's stack, and an ASTX$ directive is sufficient to return control to the task. See the *IAS Executive Facilities Reference Manual* for a description of AST service routines.

The validity of the AST entry points is not checked when the attach request is made. If an AST entry point is odd or not in the address space of the program, the program will fail when the AST occurs. IO.ATA can be used to change the AST entry point address even though a task is already attached. An AST entry point address of zero means that no AST is required.

The "unsolicited input AST" facility works only if the terminal is in "immediate processing type-ahead" mode (Section 2.2.2, mode 3). Programs that use "read-pass-all" (including tasks linked with ODT) temporarily disable "immediate processing" so that unsolicited input ASTs will not occur until a normal read has been performed.

The format for Unsolicited Character AST is:

```
QIO IO.ATT,...<[AST],[PARAMETER2][,[AST2]>
```

where:

- AST - specifies the entry point for a routine that is entered when an unsolicited character other than Ctrl/Q, Ctrl/S, Ctrl/X, or Ctrl/O is received.
- PARAMETER2 - identifies a terminal in a multi-terminal environment. It is placed into the high byte of the first word on the stack (sp+00) when the trap occurs. The low byte contains the unsolicited character.
- AST2 - specifies the entry point for a routine that is entered when an unsolicited Ctrl/C is received.

When a Ctrl/C or unsolicited input AST occurs, the task's stack will contain the following values:

- SP+16 - Event flag mask word for flags 1 to 16
- SP+14 - Event flag mask word for flags 17 to 32
- SP+12 - Event flag mask word for flags 33 to 48
- SP+08 - Event flag mask word for flags 49 to 64
- SP+06 - PS of task prior to AST
- SP+04 - PC of task prior to AST
- SP+02 - Task's Directive Status word
- SP+00 - Low byte = data;High byte = parameter

One extra parameter is put onto the task's stack, and must be removed before exiting the AST routine. See the *IAS Executive Facilities Reference Manual* for a description of AST service routines.

The validity of the AST entry points is not checked when the attach request is made. If an AST entry point is odd or not in the address space of the program, the program will fail when the AST occurs. IO.ATA can be used to change the AST entry point address even though a task is already attached. An AST entry point address of zero means that no AST is required.

The "unsolicited input AST" facility works only if the terminal is in "immediate processing type-ahead" mode (Section 2.2.2, mode 3). Programs that use "read-pass-all" (including tasks linked with ODT) temporarily disable "immediate processing" so that unsolicited input ASTs will not occur until a normal read has been performed.

If one or more parameters are omitted, the function reverts to that of Unsolicited Input Line. See Section 1.6.1 on attaching a peripheral in general. Using QIO ATT will provide the same function as QIO ATA, Unsolicited Input Line version. In timesharing systems, a terminal is not actually attached although the IO.ATT function will succeed.

If the TF.NOT subfunction is used, the function is treated as above.

This support is a conditional assembly specified by setting the parameter U$$CHA in [311,114]PARAMS.MAC.

IO.ATT (Attach Terminal)

See Section 1.6.1on attaching a peripheral in general. In timesharing systems, a terminal is not actually attached although the IO.ATT function will succeed.

When attaching to a terminal you can specify at the same time an AST (asynchronous system trap) to be entered when a complete line of unsolicited input (that is type-ahead) is entered, or, for real-time and multi-user systems only, when Ctrl/C is typed. This is done by using the function code IO.ATA instead of IO.ATT.

IO.DET (Detach Terminal)

See Section 1.6.1. Any Ctrl/C or unsolicited input AST entry point specified in Attach QIO (see above) is discarded.

IO.KIL (KIll Outstanding Requests)

All outstanding requests for this task on this terminal are aborted. The effect upon any requests depends on the terminal handler assembly parameter D$$KIL (see IAS Installation and System Generation Guide):

- D$$KIL=0 The request is terminated in the usual way, with a status of IE.ABO. If the request is for a read, any characters read so far are placed in the buffer and the number of characters read is placed in the second word of the I/O status block. Note however that no notification is given if the request has not yet been dequeued by the terminal. Thus, a program should not depend on the event flag being set or the AST occurring. For a task running at a priority less than the terminal handler (normally 248 decimal) notification will have occurred as soon as execution resumes after the directive is issued.
- D$$KIL=1 No notification is given of request termination, i.e. the I/O status block remains clear, the event flag is not set and the AST does not occur.

The former method (D$$KIL=0) is the default and is compatible with RSX-11M. The latter method (D$$KIL=1) is included for compatibility with earlier versions of RSX-11D and IAS.

Buffered output that is the result of a write request which has already been marked as done will not be affected.

IO.HNG (Disconnect (hangup) Dialup Line)

This function may be used to force a connected dialup line to disconnect.

NOTE: In some countries the line will not become free until the caller hangs up.

There are three possible error conditions:

- IE.DNR - Line not connected
- IE.IFC - Line is not a dialup line
- IE.OFL - Interface not present for line

IO.GTS (Get Terminal Support)

This function can be used to determine the facilities available in the terminal handler in use in the current system. It is fully compatible with the same function in RSX-11M.

IO.GTS takes a single parameter, which is the address of a 4-word buffer. These words are bit significant, and are set as follows:

word 1:

F1.ACR	automatic carriage-return/line feed may be supplied (TC.ACR is available)	always 1
F1.BTW	(provided for compatibility with RSX-11M)	always 0
F1.BUF	intermediate buffering available	always 1
F1.UIA	unsolicited input AST available	always 1
F1.CCO	"cancel Ctrl/O" subfunction (TF.CCO)	always 1
F1.ESQ	escape sequence recognition available	depends on E$$SEQ
F1.HLD	terminal hold mode support available	depends on H$$OLD or E$$SEQ
F1.LWC	lower case conversion available	always 1
F1.RNE	read-no-echo (subfunction TF.RNE) available	always 1
F1.RPR	read-with-prompt available	always 1
F1.RST	read-with-special terminators available	always 0
F1.RUB	character-deleting rubout on scope available	always 1
F1.SYN	terminal synchronisation support (XON/XOFF) available	always 1
F1.TRW	(Provided for compatibility with RSX-11M)	always 1
F1.UTB	(provided for compatibility with RSX-11M)	always 0
F1.VBF	(provided for compatibility with RSX-11M)	always 1

word 2:

F2.SCH	set characteristics functions available (SF.SSC, SF.SMC, SF.STT, SF.STS)	depends on S$$CHR
F2.GCH	get characteristics functions available (SG.GSC, SG.GMC)	depends on G$$CHR
F2.DCH	dump characteristics functions available (SF.SAC, SF.GAC)	depends on D$$CHR
F2.DKL	set if I/O kill aborts the current request without providing any status information	depends on D$$KIL
F2.ALT	set if altmode is echoed as "$"<CR>	depends on E$$ALT
F2.SFF	form feed can be fully simulated	depends on S$$FF
F2.CUP	(provided for compatibility with RSX-11M)	always 0
F2.FDX	(provided for compatibility with RSX-11M)	always 1

words 3 and 4 are reserved.

IO.RST (Read with Special Terminator)

This function reads characters from a TT until the input buffer is filled or any character in the ranges 0-037 or 171-177 base 8 is received.

The format of the request is:

```
QIO$  IO.RST,...,<stadd, size [,tmo]>
```

where:

- stadd - is the address of the receiving buffer.

- size - is the buffer length in bytes.
- tmo - is an optional time-out count in 10-second intervals for the full-duplex driver. If 0 is specified, no time-out can occur. Time-out is the maximum time allowed between two input characters before the read is aborted.

This support is a conditional assembly specified by setting the parameter I$$RST in [311,114]PARAMS.MAC.

IO.RTT (Read with Terminator Table)

This function reads characters from a tt until the input buffer is filled or a user-specified character (in the range 0-377) is received.

The format of the request is:

```
QIO$  IO.RTT,...,<stadd, size [,tmo], table>
```

- stadd - is the address of the receiving buffer.
- size - is the buffer length in bytes.
- tmo - is an optional time-out count in 10-second intervals for the full-duplex driver. If 0 is specified, no time-out can occur. Time-out is the maximum time allowed between two input characters before the read is aborted.

Table is the address of a sixteen-word table that specifies the end-of-read characters. Each bit in the table represents an ASCII character. The first word represents the ASCII character codes 0-17. the bits of the second word represent the ASCII codes 20-37, and so forth.

This support is a conditional assembly specified by setting the parameter I$$RST in [311,114]PARAMS.MAC.

I/O FUNCTION CODES

The I/O function codes in the table below are also available. These function codes are the "logical OR" of system macros (either the standard read logical block (IO.RLB) or the write logical block (IO.WLB)) and function codes.

Table 2–5 I/O Function Codes

I/O Function Code	System Macro	Function Code
IO.CCO	IO.WLB	TF.CCO
IO.RAL	IO.RLB	TF.RAL
IO.RNE	IO.RLB	TF.RNE
IO.WAL	IO.WLB	TF.WAL
IO.WBT	IO.WLB	TF.WBT
IO.WMS	IO.WLB	TF.WMS
IO.RNC	IO.RLB	TRF.RNC

2.6 Support of Dialup Lines

Whether a terminal is connected via a dialup line is not usually apparent to a program. The handler establishes the line and disconnects it when the caller hangs up. There are however a few special features of dialup lines on which a program can take action.

1 If a read or write request is made to a line which is disconnected or if the line becomes disconnected while a request is in progress an error status of IE.DNR is returned.

2 It is possible to force the disconnection of a dialup line by the IO.HNG function (see Section "IO.HNG (Disconnect (Hang up) Dialup Line)".

3 When a line is connected, the handler behaves as though Ctrl/C had been typed. The CLI allocated to the terminal is started, or a Ctrl/C event is declared if the CLI is already active.

4 In timesharing systems only, if a CLI is active when a line is disconnected, a Ctrl/C event is declared.

The entire process of answering a telephone line and disconnecting from it at the end of the call is controlled by the terminal handler. However it may be useful to know the exact steps involved in terms of events on the telephone line. These are summarized in Table 2–6.

Table 2–6 Handling of Dialup Lines

Event	Action
Telephone rings	Handler enters a wait period of M$$ANS seconds (note 1) during which time further ring signals are ignored. See also note 2.
End of M$$ANS time	Handler waits for another ring. If this is not seen in M$$RNG seconds, the line is dropped since the caller is assumed to have hung up.
Ring seen	The phone is answered. After a pause of M$$CAR seconds carrier is applied. The handler then waits for M$$WIC seconds for the caller to apply carrier. If this period expires the line is dropped.
Caller applies carrier	The line has been established. Transfer requests will be accepted.
Carrier is lost	Ctrl/S (XOFF) is simulated to minimize the amount of data lost on output. The handler will wait for M$$WCR seconds for carrier to be recovered. If it is, Ctrl/Q (XON) is simulated to resume output and the carrier loss is transparent to the rest of the system except possibly for a few garbled characters. See also note 3.
Carrier recovery time expires	The connection has been lost. The line becomes "not ready" again.

Notes on Table 2–6.

1 Names of the form "M$$xxx" refer to assembly parameters defined in the file PARAMS.MAC. See *IAS Installation and System Generation* Guide.

2 Lines connected via DZ11 interfaces are answered as soon as the first ring signal is seen.

3 Certain countries (for example, the UK) require that a line be dropped within a very short time of carrier loss. If the assembly parameter M$$UK is non-zero, M$$WCR is the time to wait in ticks.

2.7 Auto-Baud Detection

The Terminal Handler now can automatically detect the line speed of a terminal connected to a dial-in line.

The Terminal Handler samples the line's input character, determines the incoming caller's baud rate, and sets the interface speed accordingly.

2.7.1 Dial-in Interface

When you dial into the system, you will not receive the customary PDS WELCOME (time Sharing) message or the customary MCR display. Your terminal will display nothing at first: you must press [RETURN] several times so your baud rate can be determined. The system will then send you the customary log in banner. Note that you no longer may press [Ctrl/C] when you dial into a system: you must press [RETURN].

Previously, if the response to the USER NAME prompt was blank, PDS would hang up the dial-in line. Now if you type too many [RETURN]'s PDS will redisplay the USER NAME prompt.

2.7.2 How to Enable Auto-baud Detection

The TER command allows a user logged in under [1,1] to change the characteristics of a specified terminal. Auto-baud detection can be enabled with MCR with the command

```
MCR>TER /AUTO
```

or under PDS with the commmand

```
PDS> SET TERM:TTN AUTOBAUD
```

If you attempt to set a line to auto-baud that is not a dial-in line, you will receive the following error message:

```
<Characteristic cannot be modified for this line>
```

2.8 Escape Sequence Support

If a terminal has characteristic TC.ESQ set Table 2–2, then "escape" (octal code 33) will be treated not as a read terminator but as the start of an "escape sequence". In this case characters have special meanings. Escape sequences are used to extend the number of terminal control functions and special characters available without increasing the number of character codes.

Terminals can only be set for escape sequence recognition if the terminal handler is configured to include support for the escape sequence type required. Section 2.8.1 describes the types of escape sequence support which can conditionally be included in the terminal handler by setting the parameter E$$SEQ. The *IAS Installation and System Generation Guide* describes how to configure the handler.

2.8.1 Types of Escape Sequence Support

The effect of setting characteristics TC.ESQ and TC.ANS for a terminal depend on the type of escape sequence supported by the active terminal handler as follows:

1 No escape sequence supported by terminal handler (E$$SEQ=0)

- Terminals cannot be set to pass escape sequences.

2 VT52-type sequences only supported by terminal handler (E$$SEQ=1)

- If TC.ESQ is set for a terminal then, on input,
- escape sequence characters in Table 2–7 are translated by the handler into a single negative byte to form part of status return to a task. The codes for the negative byte values are given in column 1 of Table 2–7. If the terminal handler does not recognise the sequence, the input characters are placed exactly as issued in the user's buffer, preceded by the byte "33". The value of TC.ANS is ignored.

3 ANSI sequences only supported by terminal handler (E$$SEQ=2)

- If TC.ESQ is set for a terminal then, on input, valid ANSI sequences are passed for the terminal but are not translated. The whole input sequence is simply placed exactly as issued in the user's buffer preceded by the byte "33". The value of TC.ANS is ignored.

4 Both sequence types supported by terminal handler (E$$SEQ=3)

- If TC.ESQ is set and TC.ANS is zero the action is as described for TC.ESQ in 2 above, If both TC.ESQ and TC.ANS are set the action is as described in 3 above.

5 Both sequence types supported by terminal handler with no translation (E$$SEQ=4)

- If TC.ESQ is set and TC.ANS is zero then VT52-type sequences are passed but are not translated. The sequence is simply placed exactly as issued in the user's buffer preceded by the byte "33". If both TC.ESQ and TC.ANS are set the action is as 3 above.

2.8.2 Valid ANSI escape sequences

The handler allows the following valid sequences for a terminal with characteristic TC.ANS:

1 Escape sequences

```
ESC (IC)...(IC) (FC)
```

where:

- (IC) - is a character in the range 40 to 57 (octal) inclusive.
- (FC) - is a character in the range 60 to 176 (octal) inclusive.

2 Control sequences

```
ESC [ P(1)...P(n)I(1)...I(m)(FC)
```

where:

- [- is the bit combination 133 (octal)
- P(i) - is a character in the range 60 to 77 (octal) inclusive.
- I(j) - is a character in the range 40 to 57 (octal) inclusive.
- (FC) - is a character in the range 100 to 176 (octal) inclusive.

3 Graphic characters

```
ESC N (FC)
```

where:

- N - is the bit combination 116(octal).
- (FC) - is a character in the range 41 to 176 (octal) inclusive.

4 Further graphic characters

```
ESC O (FC)
```

where:

- O - is the bit combination 117 (octal).
- (FC) - is a character in the range 41 to 176 (octal) inclusive.

Table 2–7 Encoding of VT52-type Escape Sequences

Name of Code	Description	Sequence	Notes
ES.CUP	Cursor up	A	
ES.CDN	Cursor down	B	
ES.CRT	Cursor right	C	
ES.CLF	Cursor left	D	
ES.EGR	Enter graphics mode	F	
ES.XGR	Exit graphics mode	G	
ES.HOM	Cursor home	H,PQ	1
ES.DES	Erase (delete) to end of screen	J,PZ	1
ES.DEL	Erase (delete) to end of line	K,PX	1
ES.FFD	Forward field	R	
ES.BFD	Backward field	Q	
ES.EHS	Enter hold screen	[,PU	1
ES.XHS	Exit hold screen	\,Pu	1
ES.EKL	Enter keyboard lock	OE	
ES.XKL	Exit keyboard lock	Oe	
ES.EAL	Enter alarm mode	OG	
ES.XAL	Exit alarm mode	Og	
ES.ERV	Enter reverse video	OJ	
ES.XRV	Exit reverse video	Oj	
ES.DLD	Delete line down	ON	
ES.ILU	Insert line up	OO	
ES.SSA	Start selected area	OP	
ES.ESA	End selected area	OQ	
ES.TSA	Transmit selected area	OS	
ES.TAL	Transmit all	OV	
ES.TCC	Transmit cursor character	OW	
ES.TOB	Terminal overflow buffer	OX	
ES.SON	Terminal switched on	O{	
ES.RTS	Request to scroll	O|	
ES.SOV	Screen overflow	O}	
ES.PDL	Paragraph delimiter	PA	2
ES.TCL	Transmit cursor line	PB	2
ES.WRU	Write the ruler	PC	2
ES.DLU	Delete line up	PD	2
ES.CEM	Change emphasis	PE	2
ES.ILD	Insert line down	PF	2
ES.EIM	Enter insert mode	PI	

Table 2–7 (Cont.) Encoding of VT52-type Escape Sequences

Name of Code	Description	Sequence	Notes
ES.XIM	Exit insert mode	Pi	
ES.TMS	Transmit message	PM	2
ES.TDT	Transmit data	PN	2
ES.CJF	Clear and justify	PR	2
ES.DCH	Delete character	PS,?p	1,2
ES.CMD	Command delimiter	PT	2
ES.JFY	Justify	PV,?q	1,2
ES.ETX	Cursor to end of text	PW	2
ES.TCM	Transmit command	?s	
ES.US0	User sequence 0	0	3
ES.US1	User sequence 1	1	3
ES.US2	User sequence 2	2,?w	1,3
ES.US3	User sequence 3	3,?x	1,3
ES.US4	User sequence 4	4,?t	1,3
ES.US5	User sequence 5	5,?u	1,3
ES.US6	User sequence 6	6	3
ES.US7	User sequence 7	7	3
ES.US8	User sequence 8	8	3
ES.US9	User sequence 9	9	3
ES.CLN	Copy line	V,Pb	1,2
ES.EPC	Enter printer control	W	
ES.XPC	Exit printer control	X	
ES.CSC	Copy screen	]	
ES.EAC	Enter autocopy	^	
ES.XAC	Exit autocopy	_	
ES.PSC	Print screen	PH	2
ES.PLN	Print cursor line	PJ	2
ES.EAP	Enter auto print	PY	
ES.XAP	Exit auto print	Py	
ES.ELA	Enter linear addressing	OC	
ES.XLA	Exit linear addressing	Oc	
ES.EAT	Enter auto-tab mode	OI	
ES.XAT	Exit auto-tab mode	Oi	
ES.EAK	Enter alternate keypad	PK	
ES.XAK	Exit alternate keypad	Pk	
ES.KRC	Clear receive checksum	O[	
ES.KTC	Clear transmit checksum	O\	

Table 2–7 (Cont.) Encoding of VT52-type Escape Sequences

Name of Code	Description	Sequence	Notes
ES.KRT	Transmit receive checksum	O]	
ES.KTT	Transmit transmit checksum	O^	
ES.AIN	Initialize abort flag	O_	
ES.ATR	Transmit abort flag	O'	
ES.ANO	No output aborted	Ox	
ES.ACO	Copier aborted	Oy	
ES.APR	Printer aborted	Oz	
ES.ENT	Enter	?M	

Notes referred to by number in last column of Table 2–7:

1 Alternative sequences are shown separated by a comma. Either sequence will be accepted and will produce the corresponding translation. The first mentioned sequence will be produced on output.

2 The final letter may be in either upper or lower case.

3 The meaning of these sequences is not defined. User-written software is free to use them for any purpose.

2.8.3 Input of Escape Sequences

This section describes how the terminal handler processes escape sequence translation, on input, for non-block mode terminals. The handler translates escape sequences either when E$$SEQ=1 or when E$$SEQ=3 with TC.ANS=0 (see Section 2.8.1). The handler's actions are as follows:

1 If the sequence has a single-byte translation the request is terminated. The low byte of the first word of the I/O status block contains IS.SUC. The high byte contains the code for the escape sequence. The second word contains the number of characters placed in the buffer.

2 If there is no single-byte translation, the byte "33" is placed in the buffer followed by the characters comprising the escape sequence. The termination code IS.ESQ is placed in the first word of the I/O status block. The byte count placed in the second word includes the escape sequence.

3 If there is not enough space in the buffer to hold an escape sequence as described in (2), the current request is terminated with a status of IS.SUC. The handler starts to pass the sequence on the next read. If the buffer is still too small the sequence will be returned in sections, each of which except the last will have a status of IS.PES (partial escape sequence). The last read has a status of IS.ESQ.

4 If an invalid escape sequence is received, it is passed to the user as in (2), but the termination code is IE.IES. If the sequence has to be split as in (3) only the last section will have this error return.

For compatibility with RSX-11M and earlier versions of IAS, it is possible to force all escape sequences to be treated as in (2) (or (3) if necessary) above. If the terminal characteristic TC.CEQ ("COMPATIBLE" to the TER (MCR) or SET TERMINAL (DCL) commands) is set to 1, an escape sequence is never translated into a single byte even if such a translation exists.

2.8.4 Output of Escape Sequences

To output escape sequences you place the escape character (octal 33), followed by the constituent characters, in the buffer of a write request. Additionally, if the terminal handler is configured to support translation of VT52-type escape sequences, you can output the sequences which have a single-byte translation by using the translation as the "vertical format character" of a write QIO. In this case the escape sequence will be output after all the characters in the write buffer. If you wish to output just an escape sequence you should issue the write QIO with a buffer size of zero, but the buffer address must still be a valid address in the user's task (for example, 0).

2.9 Support of Block-Mode Terminals

This section assumes familiarity with the facilities provided by the VT61, and should be read in conjunction with the VT61 User's Manual. If a VT61 terminal is set in block mode (TC.BLK, or the BLOCKMODE option to TER or SET TERMINAL), the terminal transmits data a block at a time. A typical application for this would be a text editor, where a page of data is sent to the terminal, edited locally without involving the computer, then retransmitted to the computer, at an operator command, when editing is complete.

Programming for such a block-mode terminal is very similar to programming for a normal terminal. Output is performed in exactly the same way. Input is performed in the same way, one "line" at a time into a record-sized buffer. The handler breaks up the block of input supplied by the terminal into these smaller records, which are terminated in the usual way by carriage return or an escape sequence. A block-mode terminal must be set in escape sequence mode to function properly.

The single major programming difference occurs because a VT61 is operated in "transmit request" mode. This means that when the operator depresses one of the "enter" or "transmit" keys the terminal merely sends the corresponding escape sequence, rather than actually transmitting a block of data. It is up to the application program to detect this escape sequence (which is one of the translatable subset described in Section 2.7), by keeping a read QIO permanently in progress. The same escape sequence should then be transmitted back to the terminal, and further read QIOs issued to read the block of data. The first read will always be terminated with a status of IS.EOT, and will contain no characters. This should be ignored. The last record of the block will also be terminated with a status of IS.EOT, after which no further reads should be issued except to wait for the next "request to transmit".

It is possible, depending upon the application, that an escape sequence will occur in the middle of a "line", for example "enter reverse video mode". In this case the escape sequence will terminate a read and the remainder of the line must be read by a subsequent QIO. A particular example of this is an application which uses the "transmit command" facility of the VT61. Such a command will be prefixed by the "command delimiter" escape sequence (ES.CMD) which will result in the first read obtaining a zero length record with this terminator.

As an alternative to using normal read QIOs to read single records, the "read-pass-all" function (IO.RAL) may be used to read the entire block in one operation. In this case the specified buffer should be large enough to contain an entire block of input. If it is not, the block will be returned in sections, each of which except the last will have a termination code of IS.SUC. The last, or only, section will have a termination code of IS.EOT. For IO.RAL to work correctly, the handler must be built with a large enough node pool to contain an entire block at one time. See the description of assembly parameter N$$ODS in the *IAS Installation and System Generation Guide*.

Operation of a VT61 in forms mode (TC.FRM, FORMSMODE option) differs from the above only in the respect that the horizontal tab character will also terminate a read, with a status of IS.TAB.

There is no support for the checksum facility of the VT61. Block mode terminals made by other manufacturers are not supported unless they are compatible with the VT61. In particular:

1 They must respond to the characters XOFF and XON (octal codes 21 and 23 respectively) to suspend and resume transmission.

2 A block of data, unless it is a single escape sequence, must be preceded by STX (octal 2) and terminated by EOT (octal 4).

3 The terminal must expect every transmission from the computer (correponding to a single write QIO) to be bracketed as in 2.

4 The syntax of valid escape sequences (see Table 2-7) must be the same.

2.10 Low Speed Paper Tape Reader Support

If the assembly parameter T$$APE is non-zero, the terminal handler will support low speed paper-tape reader attachments to terminals. The paper-tape reader must respond to the characters XON (Ctrl/Q, start reading tape) and XOFF (Ctrl/S, stop reading tape). The terminal must have been set in TAPE mode using the TER /TAPE (MCR) or SET TERMINAL TAPE (DCL) commands. If there is a tape in the reader, which must be switched on, typing Ctrl/B will start reading the tape.

The character Ctrl/T, either punched on the tape or entered by turning the reader off and typing, will stop reading the tape and resume input from the keyboard. No characters read from the tape will be echoed. Prompts supplied as part of a read request (using the IO.RPR subfunction) will not be printed.

2.11 Other Supported Features

This section covers support of Parity, Character Silo, Fill Character, Other Manufacturers' Terminals, Full Duplex operation, Binary Terminals and Remote Terminals.

2.11.1 Parity Support

The terminal handler supports the hardware parity generation and detection feature of the DH11, DJ11, DL11 and DZ11 interfaces. For interfaces with settable line parameters (DH11 and DZ11), parity type (odd, even or none) is a settable characteristic. On DJ11 and DL11 interfaces, parity type is determined by jumpers or switches on the interface.

Parity generation on output for such terminals is automatic. On input, an error status of IE.VER is returned if a parity error is reported by the interface.

There is no parity support for the DC11, DL11A, DL11B or KL11 interfaces.

2.11.2 Character Silo Support

Some multiplexer interfaces (DH11, DZ11) have a dynamically variable silo alarm level. This means that they can be set to cause an interrupt only after a certain number of characters have been received. The Terminals and Communications Handbook contains a full description of this facility under the appropriate interfaces.

The terminal handler uses this facility to reduce the number of read interrupts, and their associated overhead, when a large volume of input is being received (unless the assembly parameter R$$INT is zero). This process is normally transparent to the user and the programmer alike. However, if the input rate drops sharply, for example at the end of a transmission from a high-speed block mode terminal, a short pause, never more than one second, may occur between typing characters and their being echoed. This is a normal and unavoidable part of handler operation and should be ignored.

2.11.3 Fill Characters

Some terminals, particularly older hard-copy terminals, require that some characters that require an exceptional amount of time to process, for example carriage return, be followed by "fill" characters which do not require any action. The terminal handler offers the following support for fill characters.

1 Fill after carriage return. If the characteristic TC.HFL is in the range 1-6, that same number of null fill characters will be supplied after a carriage return. If TC.HFL is equal to 7, the number of fill characters is designed to be suitable for the LA30S DECwriter, and depends on the current horizontal position.

NOTE: A fill value of 7 must NOT be used with any terminal other than an LA30S.

2 Fill after backspace and horizontal tab. If the characteristic TC.HFL is non-zero, each of these characters will be followed by a single null fill character.

3 Fill after line feed. If the characteristic TC.VFL is equal to 1, line feed will be followed by 6 null characters. This is intended to be used with VT05 terminals at 2400 baud.

4 LA36 forms feed option fill. If the characteristic TC.LVF is equal to 1, the characters vertical tab and form feed are followed by 66 null fill characters. This is intended to be used with the LA36-KV forms feed option.

2.11.4 Support of Other Manufacturers' Terminals

The terminal handler fully supports all terminals mentioned as supported in the IAS Software Product Description. The handler has been designed to be as flexible as is reasonably practical in other respects, but with the many hundreds of terminal products on the market it is not possible to provide the facilities necessary to support all of them. This applies particularly to fill character requirements (see Section 2.11.3. Further, the handler will not cope with terminals which use control characters other than in the way defined by the ASCII and ISO standards.

2.11.5 Full Duplex Operation

It is possible to operate a terminal in "full duplex" mode, in which input and output operate completely independently. This is likely to be useful, for example, if the terminal handler is used to interface to an intelligent terminal or to another computer. Full Duplex is enabled at task level by setting to 1 the characteristic TC.FDX. The corresponding terminal commands are

```
MCR>TER /FULLDUPLEX
```

or

```
PDS> SET TERMINAL FULLDUPLEX
```

In this mode, reads and writes can be performed simultaneously on the same terminal, either by the same task or by different tasks. There is no interaction at all between reading and writing, except for certain control characters (see below). Also, no echoing is performed.

In full duplex operation, the five characters Ctrl/C, Ctrl/O, Ctrl/Q (XON), Ctrl/S (XOFF) and Ctrl/X continue to have their usual significance on input. In particular, Ctrl/S and Ctrl/Q can be used by the terminal to control output from the computer in the case of buffer overflow.

2.11.6 Binary Terminals

It may be necessary to receive all characters sent by a terminal, including the special control characters Ctrl/C, Ctrl/O, Ctrl/Q, Ctrl/S and Ctrl/X. This may be done by setting the terminal in "binary" mode (characteristic TC.BIN set to 1). In this case these characters will be passed to any task which performs a read-pass-all request (sub-function TF.RAL). The terminal must also be set to "defered processing" read-ahead mode.

2.11.7 Reading Control Characters

Normally the terminal handler does not pass the control characters Ctrl/C, Ctrl/O, Ctrl/S and Ctrl/Q to a program which reads from a terminal even if the terminal is set in deferred processing mode (see Section 2.2.2).

A program can be passed some or all of these control characters when all the following conditions are satisfied:

- The terminal is set in deferred processing mode (see Section 2.2.2).
- The program performs a read-pass-all (see Section "TF.RAL").
- One (and only one) of the terminal characteristics TC.BIN, TC.CSQ or TC.CTC is set for the terminal.

The three characteristics have the following effect:

Terminal Characteristic	Characters Passed
TC.BIN	All characters.
TC.CSQ	All characters except Ctrl/S and Ctrl/Q (this is useful for applications such as terminal synchronization on a VT100 with smooth scrolling).
TC.CTC	All characters except Ctrl/C, Ctrl/S and Ctrl/Q.

2.11.8 Remote Terminals

When a terminal is connected via modems through a private line (as opposed to a switched network), it can be specified as "REMOTE" when the terminal handler is configured (see the *IAS Installation and System Generation Guide*). If the terminal is specified as "REMOTE", "Request to Send" and "Data Terminal Ready" will be set for the line when the terminal handler is loaded. The line can then be used as if it were a directly connected line.

NOTE: The special support for dialup lines described in Section 2.6 does not apply to remote lines.

2.12 The Single-Terminal Handler (TT01)

This handler supports only a single terminal (normally the console) on a DL11 line, and has very limited features.

Only the following special characters are recognized on input:

Ctrl/C	(see below)
Ctrl/U	(see Section "Ctrl/U")
Ctrl/X	(see Section "Ctrl/X")
Ctrl/Z	(see Section "Ctrl/Z")
ALTMODE	(see Section "ALTMODE")
RUBOUT	(see below)

Ctrl/C is always processed as in method 1 of Section "Ctrl/C", that is, any current input is unaffected. MCR is always invoked; there is no facility for specifying a Ctrl/C AST.

RUBOUT is always echoed as a single backslash "\" if there is a character to be rubbed out.

There is no read-ahead support. No fill characters are supplied.

Only the following function codes are supported:

IO.RLB, IO.RVB, IO.RPR, IO.WLB, IO.RAL, IO.RNE, IO.ATT, IO.DET, IO.KIL

There is no "set characteristics" function in TT01.

3 AFC11, AD01 Analog to Digital Converters

3.1 Introduction To AFC-11, AD01

The AFC-11 and AD01 devices are used for industrial and laboratory analog data acquisition. The AFC-11 is a flying multi-channel, multi-gain analog to digital (A/D) multiplexer. Under program control, the AFC-11 performs a 13-bit A/D conversion at a rate of 200 samples per second, with a maximum rate of 20 samples per second per channel. The AFC-11 is capable of multiplexing a maximum of 1024 differential input analog signals.

The AD01 is also a multi-channel A/D converter. It differs from the AFC-11 in the following respects:

- It is capable of much higher sampling rates.
- It performs 10-bit A/D conversions.
- It can multiplex up to 64 analog signals.

3.2 Functional Characteristics

The AFC-11 and AD01 handlers have two modes of operation: single-sample, or multi-sample.

3.2.1 Single-Sample Mode (Function Code IO.R1C)

In single-sample mode, both the gain and the channel number are obtained from a control word in the QIO request node. See Table 3-1. The A/D value is placed in the second word of the I/O status block. Single-mode allows quick reference to the current analog value of a given channel and eliminates the need for validating buffers.

3.2.2 Multi-Sample Mode (Function Code IO.RBC)

To function in multi-sample mode, the user must define two buffers of equal size. The first is the control buffer, which contains the control words needed to perform an A/D conversion per channel specified. See Figure 3–1.

The second buffer is used to store the results of the conversion. These results are placed in the corresponding location of the data buffer.

In multi-sample mode the user can sample many channels at approximately the same time without having to queue multiple I/O requests.

NOTE: Identical channel numbers should not be specified in the multi-sample mode when using the AFC-11. Otherwise, timing problems may result.

3.2.3 QIO System Macro Format

To initiate an A/D conversion, the user task issues a QIO request through a QIO system macro in the following format:

Single-Sample

```
QIO$ IO.R1C,lun,ef,pri,iosb,ast,<cntw>
```

Multi-Sample

```
QIO$ IO.RBC,lun,ef,pri,iosb,ast,<stadd,size,stcnta>
```

For Single-sample mode, cntw specifies the control word (see Figure 3–1. For Multi-sample mode, the parameter words have the following significance.

- stadd - Address of the start of the data buffer, relative to the user task. The address must start on a word boundary.
- size - The size of both the data buffer and the control buffer in bytes. The size must be an even number of bytes.
- stcnta - Address of the start of the control buffer relative to the user task. Each word of the control buffer must be set up as shown in Figure 3–1. The address must start on a word boundary.

The control word specifies the channel to be sampled and the gain value to be applied.

As shown in Figure 3–1, this control word is provided either as a parameter in the DPB of the QIO request or in a control buffer depending on whether single-sample mode or multi-sample mode is being used.

The maximum number of channels present on the configuration is specified at system generation. This value is specified in the AFC-11 or AD01 entry in the physical unit directory (PUD). If the value specified for the channel number in the control word is greater than that stored in the physical unit directory, the AFC-11 or AD01 handler terminates the I/O request and returns an appropriate error status.

User requests for time based sampling of a particular channel(s) should be made as follows.

1 The user task queues a sample request to be performed for one or a series of channels. The channel number(s) and gain(s) must be specified in the control word, as indicated in Figure 3–1.

2 The user task then issues a Mark Time directive for at least three clock ticks to avoid skewing effects before queuing another sample request. Thus, there is a 20 sample per second per channel restriction on the sampling rate.

3 If an A/D error occurs, the I/O request is terminated and the appropriate error code is placed in the first word of the user status buffer. The second word in the status buffer contains the number of valid samples taken prior to the error. Values for prior samples will be found in the data buffer, as expected.

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 positive full scale reading is imminent and to change to a lower gain whenever the last A/D value recorded was less than half full scale. This method maintains maximum resolution while avoiding saturation.

Figure 3–1 A/D Conversion Control Word

Bit	Meaning					
0 – 10	Channel number: 0 – 1023 (AFC) 0 – 63 (AD01)					
12 – 15	Gain value for this sample, expressed in the bit patterns shown as follows:					
	Bits				AFC Gain	AD01 Gain
	15	14	13	12		
	0	0	0	0	1	1
	0	0	0	1	1	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

3.2.4 AFC/AD01 Status Returns

The lowest byte of the I/O status block contains a code indicating the disposition of the QIO request. These status return codes for the AFC/AD01 handler are shown below.

Symbol	Meaning
IS.SUC	Successful completion
IE.BAD	Bad parameter (illegal channel)
IE.IFC	I/O function not recognized
IE.DNR	A/D timeout on sample
IE.SPC	Illegal buffer address or count
IE.PRI	Privilege violation

See Appendix A for a complete list of I/O status returns.

4 Disk Handlers

4.1 Disk I/O Handlers

Table 4–1 contains a brief summary of the characteristics of the disks supported by the disk handler tasks and relates the devices to the handler tasks that service them. A complete description of each disk is provided in the *PDP-11 Peripherals Handbook*.

Table 4–1 Standard Disk Devices

Controller/ Drive	Installed Task Name	RPM	Secs	Trks	Cyls	Bytes/ Drive	Decimal Blocks
RF11/RS11	DF....	1800	–	1	128	524,288	1024
RHxx/RS03	DS....	3600	64[1]	1	64	524,288	1024
RHxx/RS04	DS....	3600	64[1]	1	64	1,048,576	2048
RP11E/RP02	DP....	2400	10	20	200	20,480,000	40,000
RP11C/RP03	DP....	2400	10	20	400	40,960,000	80,000
RH11/RM02	DR....	2400	32	5	823	67,420,160	131,680[6]
RHxx/RM03	DR....	3600	32	5	823	67,420,160	131,680[6]
RHxx/RP04,RP05	DB....	3600	22	19	411	87,960,576	171,798
RHxx/RP06	DB....	3600	22	19	815	174,423,040	340,670
RP07	DB....	3600	50	32	630[5]	516,096,000	1,008,000
RM80	DR....	3600	31	14	559[5]	124,214,272	242,606
RH70/RM05	DR....	3600	32	19	823	256,196,608	500,384
RK11/RK05	DK....	1500	12	2	200	2,457,600	4800
RL11/RL01	DL....	2400	40[2]	2	256	5,242,880	10,240[6]
RL11/RL02	DL....	2400	40[2]	2	512	10,485,760	20,480[6]
RK611/RK06	DM....	2400	22	3	411	13,888,512	27,126[6]
RK611/RK07	DM....	2400	22	3	815	27,810,800	53,790[6]
RX11/RX01	DX....	360	26[3]	1	77	256,256	494
RX211/RX02	DY....	360	26[4]	1	77	512,5124	9884
RA60	DU....	3600	42	4	2382	204,890,112	400,176

1 The RS03 has 64 words per sector; the RS04 has 128 words per sector.
2 The RL01 and RL02 each have 128 words per sector.
3 The RX01 has 64 words per sector.
4 These numbers are for a double-density diskette. The RX02 has 128 words per sector when formatted as double density and 64 words per sector when formatted as single density.
5 The RP07 and the RM80 each have two additional CE cylinders.
6 The last physical track on the pack is reserved for recording bad sector locations. Thus the number of blocks available to the user is reduced by the number of sectors in a track.

Table 4–1 (Cont.) Standard Disk Devices

Controller/ Drive	Installed Task Name	RPM	Secs	Trks	Cyls	Bytes/ Drive	Decimal Blocks
RA70	DU....	4000	-	-	-	280,084,992	547,041
RA80	DU....	3600	31	14	546	121,325,568	236,964
RA81	DU....	3600	51	14	1248	456,228,864	891,072
RA82	DU....	3600	-	-	-	622,932,480	1,216,665
RA90	DU....	3600	-	-	-	1,216,590,336	2,376,153
RC25	DU....	2850	31	2	796	26,061,824	50,902
RD31	DU....	-	-	-	-	21,278,720	41,560
RD32	DU....	-	-	-	-	42,600,448	83,204
RD51	DU....	3600	16	4	306	10,027,008	19,584
RD52	DU....	-	-	-	-	30,965,760	60,480
RD53	DU....	-	-	-	-	71,000,064	138,672
RD54	DU....	-	-	-	-	159,334,400	311,200
RX50	DU....	300	10	1	80	409,600	800
RX33	DU....	360	15	160	615	1,228,800	2,400

1 The RS03 has 64 words per sector; the RS04 has 128 words per sector.
2 The RL01 and RL02 each have 128 words per sector.
3 The RX01 has 64 words per sector.
4 These numbers are for a double-density diskette. The RX02 has 128 words per sector when formatted as double density and 64 words per sector when formatted as single density.
5 The RP07 and the RM80 each have two additional CE cylinders.
6 The last physical track on the pack is reserved for recording bad sector locations. Thus the number of blocks available to the user is reduced by the number of sectors in a track.

To optimize the moving head disk handler operations, nondata transfer QIOs and data transfer QIOs are dequeued according to priority, but they are processed differently. The nondata transfer QIOs are dequeued, processed and returned immediately to the user task, while data transfer QIOs are dequeued but are not necessarily processed immediately.

When the disk is ready to perform a data transfer, the disk handler selectively processes the list of dequeued data requests according to the following rules.

1 The highest priority request at that moment is processed first regardless of the current disk head position.

2 Requests with the same priority and from the same task are not always processed in the order which they were queued. For example the DB and DR handlers do not process requests on a first-in/first-out (FIFO) basis. You should not rely on any specific order of processing I/O requests of the same priority.

3 Requests with the same priority, but requested by different tasks are selectively processed according to the relative destination of the disk head for the requests as compared to the current disk head position. Rule 2 above is also considered in this section.

Nondata and data QIOs are processed asynchronously.

The following techniques can be used to control processing in a program containing both data and non-data QIOs.

1 If the order of processing is important to the program, WAITFOR or STOPFOR directives can be used to ensure that the desired QIO is processed before further QIOs are issued.

2 Nondata transfer QIOs should use an event flag different from the event flag used by data transfer QIOs when both are contained in the same program. This ensures that a WAITFOR directive associated with a data transfer QIO is not affected by the setting of an event flag from a nondata transfer QIO.

4.1.1 RS03 Fixed-Head Disk

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

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

The RP11 controller/RP02 or RP03 disk pack 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.

4.1.2 RM02/RM03/RM05/RM80 Disk Pack

The RM02/RM03, RM05, and RM80 are MASSBUS disk drives and adapters that use the existing MASSBUS controller. With a single head per surface, they provide a 1.2-Mb/s data transfer rate. PDP-11/70 systems use the RM03, RM05, and RM80 with the RH70 controller on PDP-11/70 systems. All other systems use the RM02 with the RH11 controller.

4.1.3 RP04, RP05, RP06, and RP07 Disks

The RP04 or RP05 (RH11-RH70 controller/RP04 or RP05 disk packs) disk packs consist of 19 data surfaces and a moving read/write head. Both offer large storage capacity with rapid access time. The RP06 disk pack has approximately twice the capacity of the RP04 or RP05. The RP07 fixed-media disk has approximately three times the capacity of the RP06.

4.1.4 RK11/RK05 or RK05F Cartridge Disks

The RK11 controller/RK05 DECpack cartridge disk is for medium-volume, random-access storage. The removable disk cartridge offers the flexibility of large offline capacity with rapid transfers of files between online and offline units without necessitating copying operations. The RK05F has twice the storage capacity of the RK05 and has a fixed (nonremovable) disk cartridge.

4.1.5 RL11/RL01 or RL02 Cartridge Disk

The RL01 is a low-cost, single-head-per-surface disk with a burst data transfer rate of 512-Kb/s. The storage capacity of the RL02 is twice that of the RL01.

4.1.6 RK611/RK06 or RK07 Cartridge Disk

The RK611 controller/RK06 catridge disk is a removable, random-access, bulk-storage system with three data surfaces. The storage capacity is 6,944,256 words per disk pack. The system, expandable to eight drives, is suitable for medium to large systems.

The RK611 controller/RK07 cartridge disk is generally similar to the RK611/RK06, except storage capacity is increased to approximately 13,905,400 words per disk pack. Both RK06 and RK07 disks can use the same RK611 controller; mixing RK06 and RK07 disks on the same controller is permitted.

4.1.7 RX11/RX01 Flexible Disk

The RX11 controller/RX01 flexible disk is for low-volume, random-access storage. Data is stored in twenty-six 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 ar 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 1 and provides access to 494 logical blocks.

4.1.8 RX211/RX02 Flexible Disk

The RX211 controller/RX02 flexible disk is for low-volume, random-access storage. It is capable of operating in either an industry- standard, single-density mode (as stated for the RX11/RX01 flexible disk), or a double-density mode (not industry standard). In the single- density mode, each drive can store data exactly as stated in Section 4.1.7. In the double-density mode, data is stored in twenty-six 128-word sectors per track; there are 77 tracks per disk. The RX211/RX02 operating in the single-density mode can read disks written by an RX11/RX01 flexible disk system. In addition, disks written by the RX211/RX02 operating in the single-density mode can be read by the RX11/RX01 flexible disk system.

4.1.9 KDA50, UDA50/RA60/RA80/RA81 Disks

The KDA50 or UDA50 controller is an intelligent disk controller that contains a high-speed microprogrammed processor capable of performing all disk functions, including data handling, error detection and correction, and optimization of disk drive activity and data transfers. The controller optimizers disk activity by reording QIO$s. Therefore, QIO$ macros may not complete in the order in which they were issued. The types of drives that can be connected to the KDA50 or UDA50 controllers are the RA60 disk drive, which has a removable disk pack, and the RA80, RA81, RA82, and RA90 all of which are fixed media drives. (For data capacities and rates, see Table 4–1.) Up to four of these drives can be connected to a KDA/UDA, in any desired combinaation.

The KDA/UDA controller can perform an extensive self-test on power-up or initialization.

4.1.10 RC25 Disk Subsystem

The RC25 disk subsystem consists of a fixed-media drive and a removable-media drive, both of which revolve on the same spindle and share the same head mechanics. Each drive is a logical unit, so each RC25 disk subsystem consists of two logical units.

The RC25 Subsystem combines, in one package, a controller and a single disk drive that has a removable disk and a fixed disk. These disks reside in the drive as two separate logical units on a single spindle. Their size is the same. Both are single 8-inch disks with two surfaces, and both disks have the same data capacity. But mechanically they are different: One is a removable front-loading cartridge disk, while the other cannot be removed from the drive. The drive contains loadable Winchester heads.

RC25 subsystems are available in two types: a master drive that contains its own controller, and a slave drive, which must be connected to an RC25 master drive. Each RC25 master drive can support one RC25 slave drive. The added-on disk drive is a slave to the disk subsystem that has the controller. A master-slave configuration would contain four logical units.

4.1.11 RD31 Fixed 5.25-Inch Disk

The RD31 disk drive is a 5.25-inch fixed disk with Winchester-type heads. The RD31 is soft sectored and field formattable. The maximum capacity of the RD31 is 20 Mb.

4.1.12 RX33 5.25-Inch Half-Height Disk

The RX33 disk drive is a half-height, 5.25-inch single flexible disk. It operates as a dual speed, double-sided, diskette drive and has a maximum capacity of 1.2 Mb. The RX33 requires the RQDX3 disk controller, supports RX33 formatting, and can perform read/write operations for both RX33 and RX50 diskettes.

4.1.13 RD51 Fixed 5.25 Disk/RX50 Flexible 5.25 Disk

This subsystem consists of a hard disk (RD51) and flexible disk (RX50) combination, and a RQDX1/RQDX2 controller. In combination, they are a mass-storage medium for small systems. The basic configuration for this subsystem is an RD51 fixed-disk drive and an RX50 flexible, dual-disk drive, or both. The RX50 dual disk is addressed as two separate units resulting in a basic configuration of three disk units. Also, you can add another RD51 to increase storage capacity. Some of the characteristics of the RD/RX drives are given in Table 4–1 and in the following paragraphs.

The RD51 disk drive is a 5.25-inch fixed disk with Winchester-type heads. It has two disks with four data surfaces. The RD51 is soft sectored and field formattable. The headers for each sector contain the sector's cylinder number, head number, and sector number. The sector number is the logical sector number (0-15) that reflects the sector interleave of the disk.

The RX50 dual diskette drive is a compact, mass-storage drive with two access slots. Each slot can hold a single-sided 5.25-inch flexible disk. These diskettes are firm sectored and are not field formattable. Every track has sectors numbered from 1 to 10. The two diskettes share the same head transport mechanism.

4.1.14 RD52 Fixed 5.25-Inch Disk

The RD52 disk drive is a 5.25-inch fixed disk with Winchester-type heads. The RD52 is soft sectored and field formattable. The maximum capacity of the RD52 is 31 Mb.

4.1.15 RD53 Fixed 5.25-Inch Disk

The RD53 disk drive is a 5.25-inch fixed disk with Winchester-type heads. The RD53 is soft sectored and field formattable. The maximum capacity of the RD53 is 71 Mb.

4.1.16 RD54 Fixed 5.25-Inch Disk

The RD54 disk drive is a 5.25-inch fixed disk with Winchester-type heads. The RD54 is soft sectored and field formattable. The maximum capacity of the RD54 is 159 Mb.

4.2 Function Codes

I/O requests serviced by disk handlers are issued using the QIO$ system macro. This section describes standard and device specific QIO functions for disk handlers.

4.2.1 Standard QIO Functions

Section 1.7 discusses standard function codes for mass storage devices in general. The following sections provide details of these functions codes when used for disk devices.

READ/WRITE Logical Functions

```
QIO$ fc,lun,ef,pri,iosb,ast,<stadd,size,comp,blkh,blkl>
```

fc can have one of the following values:

- IO.WLB - Write logical block
- IO.RLB - Read logical block

The five parameter words bracketed by left and right angle brackets must be specified and must be delimited by the angle brackets. They have the following meanings:

Parameter	Meaning
stadd	Virtual starting address in memory of user's buffer for data input or output (must be on a word boundary and in some cases (for example, RP03) an even word boundary).
size	Size of the data buffer in bytes. The size must be even and non-zero. For RP03 disks it must also be a multiple of 4 bytes.
comp	0 (retains compatibility with non-mass storage carriage control logical read/write functions).
blkh,blkl	Block-high, block-low. Double precision number indicating the first logical block address on the disk specifying where the transfer starts. The value of the block number is multiplied by 256 to locate the proper disk word (for example, block number 3 means disk word 768).

ATTACH/DETACH Functions

```
QIO$ fc,lun,ef,pri,iosb,ast
```

fc can have one of the following values:

- IO.ATT - Attach disk unit
- IO.DET - Detach disk unit

4.2.2 Device-Specific QIO Functions

Additional function codes are provided by some disk handlers to support functions available for specific disk types. Table 4-3 lists these functions and shows the disks for which each is available.

Table 4–2 Device-Specific Functions for Disks

Format	Function	Disk
QIO$C IO.RPB,...,<stadd,size,,,pbn>	Read physical block	RX01,RX02 RL01,RL02
QIO$C IO.SEC,...	Sense diskette characteristics	RX02
QIO$C IO.SMD,...,<density,,>	Set media density (format diskette)	RX02
QIO$C IO.WDD,...,<stadd,size,,,pbn>	Write physical block (with deleted data mark)	RX01, RX02
QIO$C IO.WPB,...,<stadd,size,,,pbn>	Write physical block	RX01,RX02 RL01,RL02

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 (no validation will occur).
- density - is the media density as follows:

  0 = single (RX01-compatible) density

  2 = double density

4.3 Disk Status Returns

The lowest byte of the I/O status block contains a code indicating the disposition of the QIO request. These status return codes for the disk handlers are symbolized as shown below:

Symbol	Meaning
IE.BBE	Bad sector flag set in sector header.
IE.BLK	Logical block number too large.

Symbol	Meaning
IE.DAA	Device already attached.
IE.DNA	Device not attached (detach failed).
IE.DNR	Device not ready.
IE.FHE	Fatal hardware error.
IE.DNR	Device not ready.
IE.IFC	Invalid function code (access violation).
IE.OVR	Illegal overlay request.
IE.PRI	Privilege violation.
IE.SPC	One of: • Part of buffer out of user's address space. • User buffer for RP02/03 disk transfer is not on an even word boundary. • No UMRs available.
IE.SRE	Send/receive failure.
IE.VER	Parity error on device (irrecoverable error).
IE.WLK	Device write protected.

See Appendix Afor a complete list of I/O status returns.

4.4 UNIBUS Mapping Registers

All DMA devices on the UNIBUS of a PDP-11/44, a PDP-11/84, or a PDP-11/70 use UNIBUS mapping registers (UMRs) to perform DMA transfer if the machine is running in 22-bit mode. Refer to the appropriate PDP-11 Processor Handbook.

All disk handlers use the standard handler library routines to allocate, load, and deallocate UMRs. UMRs can either be statically preallocated during the initialization (that is, initial handler loading) or dynamically allocated when a transfer is requested.

The overlapped seek version of the DK disk handler (DKOVL) statically preallocates eight UMRs during initialization and keeps them until the handler exits. This handler requests eight UMRs to enable a maximum data transfer of 32K words.

All other disk handlers do not attempt to preallocate the maximum number of UMRs at initialization. Each handler preallocates only one UMR. This enables all transfers of up to 4K words to occur with no dynamic allocation overhead and does not tie up UMRs unnecessarily. If a transfer of more than 4K words is required, the handler attempts to allocate sufficient UMRs for the transfer and releases them after the transfer. If sufficient UMRs are not available for a transfer, the transfer will not occur and an error will be returned to the calling task.

Handlers which use this dynamic mechanism are:

DB.... when running on a PDP-11/44 processor
DK.... non overlapped seek version
DL....
DM....
DP....
DR.... when running on a PDP-11/44 processor
DS.... when running on a PDP-11/44 processor

DY....

Since you can SAVE a system with handlers loaded which are not using UMRs and transport the system to a processor where UMRs are required, the handlers also attempt to acquire UMRs at system start up and at power recovery.

In both cases, if the handler fails to acquire sufficient UMRs, it declares itself nonresident and exits.

4.5 Error Recovery in DB, DM and DR Handlers

Disks serviced by the DB, DM and DR handlers have Error Correction Code (ECC) facilities which are used by the handlers when necessary on disk reads. If a read error cannot be corrected through ECC or the error is on a write to disk, the handlers go through a process of trying again. This involves several attempts with the head in its current position, and if necessary recalibration and further tries. On read errors, the handlers will attempt the read by using the track offset facility of the drives. Track offset is the movement of the read heads to a given distance from the cylinder center line, which is the optimum position adopted by the heads after a seek. In the DB and DM handlers, three offset positions are tried on each side of the center line; in the DR handler, one offset position is tried on each side of the center line. At each offset position, two attempts are made to read the data. If all read attempts fail, a "hard" error is declared. The DB handler tries the read a maximum of 6+(6*2) or 18 times. The DM handler tries a maximum of 16+(6*2) or 28 times. The DR handler tries a maximum of 16+(2*2) or 20 times.

4.6 Characteristics Words for Disk Devices

Section 1.8.1 describes the general use of the four characteristics words in the PUD.

The format of U.C2 and U.C3 for disks is described in the following sections.

4.6.1 Characteristics Word 2

Word 2 (U.C2) has four fields:

1	Bits 0-3	Settable Flags
2	Bits 4-7	Fixed Flags
3	Bits 8-12	Device Type
4	Bits 13-15	Device Dependant Information

The "settable flags" field is reserved for flags which may be changed dynamically while the system is running. These are:

bit 0	U2.WCK	unit is to have read-after-write checking performed
bits 1-3		reserved

The "fixed flags" field is reserved for flags which are set up as part of System Generation and do not vary. These are:

bit 4	U2.MOH	device has moving heads
bit 5	U2.RMV	device has removable volumes
bit 6	U2.BAD	device has factory-supplied bad block information in the last track

bit 7 reserved

The "device type" field is a 5-bit field whose defined values are given in Table 4–3, column 1.

The "device dependant information" field is reserved for use by device handlers. Its layout depends on the device type. See Table 4–3, column 3.

The symbolic names above are defined in the file [1,1]EXEC.STB.

Table 4–3 Characteristics Word 2 (U.C2), Bits 8-15

Bits 8-12	Device Type	Meaning of Bits 13-15
0	unspecified	
1	RF	(number of platters)-1
2	RK11	0 unit is RK05 1 unit is RK03 2 unit is RK05F
3	RP11	2 controller is RP11C 4 unit is RP03
4	RP04/5/6	1 unit is RP04 or RP05 2 unit is RP06 (changed dynamically by handler).
5	RS	0 unit is RS03 1 unit is RS04
6	RK06/7	0 unit is RK06 1 unit is RK07
7	RX01	reserved
10	DECtape	reserved
11	RM03/02/05	0 unit is RM02 1 unit is RM03 2 unit is RM05
12	RL01/2	0 unit is RL01 1 unit is RL02
13	TU58	reserved
14	RX02	reserved

4.6.2 Characteristics Word 3

For all disk devices except RX02, word 3 (U.C3) has two fields:

- Low byte - number of sectors per track.
- High byte - For moving head disks, number of tracks per cylinder. For fixed head disks this byte is zero.

For RX02 disk devices word U.C3 contains the maximum logical block number.

5 UDC-11 Handler

5.1 Introduction to UDC-11

The UDC-11 handler task provides an interface to the PDP-11 Universal Digital Controller (UDC-11) front-end devices. The UDC-11 is a single-unit device whose name is UD.

Due to the generality of this device, a prebuilt handler task is not supplied to customers. A descriptor program (source file) describing a particular UDC-11 module configuration must be prepared, assembled, and the resultant OBJ file included in the task builder input to produce a UDC handler task for a given installation.

5.2 Source File Macros

The source file consists of a series of macros, one for each module type to be supported. Each macro takes two arguments:

1 the relative module number of the first module of a type, and

2 the number of modules of the type.

The module number specifies a hardware module position, and effectively defines an external page address. Module number n corresponds to address 171000+2n for n=0 through n=251. All modules of a type must have consecutive module addresses. The macro names for each module type are as follows:

- UDC$AO - Analog Output
- UDC$CI - Contact Interrupt
- UDC$CS - Contact Sense
- UDC$DL - Latching Digital Output
- UDC$SS - Single Shot Digital Output
- UDC$TC - Timer or Counter Modules
- UDC$AD - ADU01 A/D Converter

In addition to the above macros, a macro called UD$END must be included as the last macro in the source to mark the end of the module definition table being assembled.

5.2.1 Macros Referenced by .MCALL

Two additional macros named UD$DST (dispatch table) and UD$MDT (module description table) must be referred to by the .MCALL assembler declaration.

For example, the following module description program (file UDTB.MAC;n) is assembled to produce the object module UDTB.OBJ;n.

```
; "PROGRAM" TO CONFIGURE AN IAS HANDLER TASK TO
; SUPPORT THE FOLLOWING UDC MODULES.
;
; ADR   TYPE    FUNCTION
;
; 171000  W733  CONTACT INTERRUPT POINTS 0-15
; 171002  W733  CONTACT INTERRUPT POINTS 16-31
; 171004  W731  CONTACT SENSE POINTS 0-15
; 171006  W731  CONTACT SENSE POINTS 16-31
; 171010  M803  DIGITAL OUTPUT LATCHING POINTS 0-15
; 171012  M805  DIGITAL OUTPUT LATCHING POINTS 16-31
; 171014  M685  DIGITAL OUTPUT LATCHING POINTS 32-47
; 171016  M807  SINGLE SHOT DIGITAL OUTPUT PTS 0-15
; 171020  M687  SINGLE SHOT DIGITAL OUTPUT PTS 16-31
; 171022  W734  TIMER MODULE
; 171024  A633  ANALOG OUTPUT CHANNELS 0-3
;
  .MCALL  UD$DST,UD$MDT
;
  UDC$CI  00.,2  ;TWO CONTACT INTERRUPT MODULES
  UDC$CS  02.,2  ;TWO CONTACT SENSE MODULES
  UDC$DL  04.,3  ;THREE DIGITAL OUTPUT
           ;LATCHING MODULES


  UDC$SS  07.,2  ;TWO SINGLE SHOT DIGITAL
           ;OUTPUT MODULES
  UDC$TC  09.,1  ;ONE TIMER/CLOCK MODULE
  UDC$AO  10.,1  ;ONE ANALOG OUTPUT MODULE
  UD$END        ;END OF MODULE DEFINITION TABLE
;
  .END          ;END OF ASSEMBLY
```

5.2.2 Creating an Installation-Specific UDC Handler Task

The object module UDTB.OBJ;n is created by assembling [11,14]UDSYMBOLS.MAC, [11,14]UDMACDEFS.MAC, and UDTB.MAC.

UDTB.MAC is the user module description file, and the other files contain system symbol and macro definitions (stored on UIC [11,14] of the system disk delivered).

It is linked with appropriate library modules using the task builder to create an installation-specific UDC handler task. The following task builder commands are used to build an image of a task named UD.TSK;n.

```
[11,1]UD.TSK/PR/-AB/-FP/-FX,LP:=UDTB
[1,1]UDLIB.OLB/LB,[1,1]EXEC.STB
/
TASK=UD....
STACK=1
PAR=GEN
PRI=246
UIC=[1,1]
//
```

5.3 Interrupt/Noninterrupt UDC Modules

Noninterrupting UDC modules can be set and/or sensed by any task.

Interrupting UDC modules are divided into three classes:

1 digital point (contact interrupt) modules,

2 timer modules, and

3 A/D modules (ADU01).

All interrupts of the first two classes are serviced by a single task.

NOTE: Functional separation of digital points and time measurements is application dependent. Therefore, the UDC handler task allows interrupts to be handled by a non-privileged user-written task.

A task can connect to either (or both) of the first two classes of interrupts by providing a circular buffer to receive interrupt information, and an event flag number to allow triggering of the task whenever a buffer entry is made. The third class is synchronous (demand only) and so can be handled in a manner similar to the noninterrupt modules.

5.4 Function Descriptions

The following paragraphs describe the functions supported by the UDC-11 handler task in terms of an assembly language interface.

5.4.1 Analog Output - A633 Modules

There are four analog output channels per A633 module. The channels are numbered from zero starting with the first channel on the first analog output module.

To set an indicated channel to the indicated voltage, issue the following QIO macro:

```
QIO$   IO.SAO,lun,ef,pri,iosb,ast,<ocn,ovr>
```

where:

- ocn - Output channel number
- ovr - Output voltage representation.

The output voltage varies linearly with the binary output to the channel where values from zero to plus ten volts (+10v.) may be represented by integers from 0 to 1023.

5.4.2 Single-Shot Digital Output - M687 and M807 Modules

There are sixteen 1-shot digital output points per module. The points are numbered from zero starting with the first point on the first module.

To pulse an indicated output point, issue the following QIO macro:

```
QIO$   IO.SSO,lun,ef,pri,iosb,ast,<opn>
```

where:

- opn - Digital output point number.

To pulse a set of up to sixteen points, issue the following QIO macro:

```
QIO$   IO.MSO,lun,ef,pri,iosb,ast,<opn,mas>
```

where:

- opn - First digital output point number
- mas - 16-bit mask

Bit n of the mask corresponds to point number (opn) +n, (n=0-15). For every bit in the mask that is set, the corresponding point is pulsed.

5.4.3 Latching Digital Output - M685, M803 and M805 Modules

There are sixteen latching digital output points per module. The points are numbered from zero starting with the first point on the first module.

To set an indicated digital output point to an indicated logical value, issue the following QIO macro:

```
QIO$   IO.SLO,lun,ef,pri,iosb,ast,<opn,pp>
```

where:

- opn - Digital output point number
- pp - Logical value (point polarity)

A logical value of .TRUE. implies that contacts are closed and is represented by a word with all bits set (-1). A logical value of .FALSE. implies that contacts are open and is represented by a word with all bits cleared (+0).

To open or close a set of up to sixteen points, issue the following QIO macro:

```
QIO$   IO.MLO,lun,ef,pri,iosb,ast,<opn,pp,dp>
```

where:

- opn - First digital output point number
- pp - 16-bit mask
- dp - Data pattern

Bit n of the mask/data pattern corresponds to point number (opn) +n (n=0-15). If a bit in the mask is set, the corresponding point is opened/closed depending on the corresponding bit in the data pattern being clear/set. If a bit in the mask is clear, the corresponding point is left unaltered.

5.4.4 Contact Sense Digital Input - W731 and W733 Modules

There are sixteen digital input points per module. The points are numbered from zero starting with the first point on the first module.

To read an indicated digital input point and return the data in the second word of a specified I/O status block, issue the following QIO macro:

```
QIO$   IO.SCS,lun,ef,pri,iosb,ast,<ipn>
```

where:

- ipn - Digital input point number

The second word of the I/O status block is set to -1 if the indicated point is .TRUE. (contact closed), or to zero if the point is .FALSE. (contact open).

To read a field of sixteen digital input points and return the data in the second word of an I/O status block specified in the QIO DPB issue the following QIO macro:

- QIO$ IO.MCS,lun,ef,pri,iosb,ast

5.5 Contact Interrupt Digital Input - W733 Modules

Digital input from contact interrupt modules is reported in a requester-provided circular buffer. Each buffer entry is five words long and is of the following format:

- word 00 - Entry existence indicator
- word 01 - Change of state (COS) indication
- word 02 - Module data (current point values)
- word 03 - Module number (module interrupted)
- word 04 - Generic code (interrupting module)

The entry existence indicator is set nonzero when a buffer entry is made. When a requester has removed or processed an entry, it must clear its existence indicator in order to free the buffer entry position. Entries are made in a circular fashion, starting at the top (low address), filling in order of increasing memory addresses to the bottom (high address), and wrapping around from bottom to top. If input data 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, and a negative value is the two's complement of the number of times data has been discarded between entries. Word zero of the buffer is used by the handler task as a pointer into the buffer where the next set of interrupt information is to be entered. It is expected that the connected task will maintain its own pointer to that location in the buffer where it is to next retrieve contact interrupt data. When a task is triggered by the handler, 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 handler has entered into the buffer. The pointer maintained by the handler is to be thought of as a FORTRAN index into the buffer, i.e., the first location of the buffer is associated with the number (index) 1. The second location associated with 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.

5.5.1 Change of State (COS) Output

The change of state (COS) output indicates which point(s) of the module have changed state. The bit position of an on-bit in the COS output word provides the low order bits (3-0) of a point number and the module number, word 3, provides the high order bits (15-4).

The module data bits indicate the logical value (polarity) of each point of the module.

5.5.2 Contact Interrupt Functions Connect/Disconnect

Contact interrupt input is reported only to one task. This interrupt is controlled by two UDC handler task functions:

1 CONNECT a buffer for contact interrupt digital input, and

2 DISCONNECT a buffer from contact interrupt digital input.

To connect a buffer for contact interrupt digital input, issue the following QIO macro:

```
QIO$   IO.CCI,lun,ef,pri,iosb,ast,<buf,size,tevf>
```

- buf - Virtual address of top of buffer
- size - Length of buffer in bytes
- tevf - Trigger event flag number

If the connection is successful, the second word of the I/O status block contains .BYTE a,b where a is the number of words passed per interrupt and b is the initial FORTRAN index into the top of the buffer.

To disconnect a buffer from contact interrupt digital input, issue the following QIO macro:

```
QIO$   IO.DCI,lun,ef,pri,iosb,ast
```

To read contact interrupts statically, issue the following QIO macro:

```
QIO$   IO.RCI,lun,ef,pri,iosb,ast,<mn>
```

where:

- mn - Relative point number.

5.6 Timer (Counter) - W734 Module

Counter modules are treated in a manner similar to contact interrupts. They can be read, connected to, and disconnected from interrupts. Counter interrupt information is reported in a requester provided circular buffer which is handled in a similar manner to the contact interrupt buffer. Each entry is four words long and is of the following format:

- word 00 - Entry existence indicator
- word 01 - Module data
- word 02 - Module number
- word 03 - Generic code

To connect a timer, issue the following QIO macro:

```
QIO$   IO.CTI,lun,ef,pri,iosb,ast,<buf,size,tev,arv>
```

where:

- buf - Virtual address of top of buffer
- size - Length of buffer in bytes
- tev - Trigger event flag number
- arv - Adr of table of initial/reset values.

The buffer of initial/reset values is used to load the timers or to connect and reload them on interrupt (overflow). The buffer contains one word for each timer module. The contents of the first word (negative count) is used for the first module, etc. If a timer has a nonzero value at interrupt time, it is not reloaded, so that self-clocking modules and modules that interrupt on half count can continue incrementing from the initial value.

To disconnect a timer, issue the following QIO macro:

```
QIO$   IO.DTI,lun,ef,pri,iosb,ast
```

To read a timer, issue the following QIO macro:

```
QIO$   IO.RTI,lun,ef,pri,iosb,ast,<mn>
```

where:

- mn - Module number

To initialize a timer, issue the following QIO macro:

```
QIO$   IO.ITI,lun,ef,pri,iosb,ast,<mn,ic>
```

where:

- mn - Module number
- ic - Initial count

The value of the counter module is returned in the second word of the I/O status block.

5.7 Analog/Digital Converter - ADU01

There are eight analog input channels per ADU01 module. The channels are numbered from zero starting with the first channel on the first module. Except for timer and contact interrupt processing, no other UDC function is processed during A/D sampling.

The QIO calls for ADU01 service are identical to those used for AFC-11. See Chapter 3 for the macro forms of the READ SINGLE A/D POINT (IO.R1B) and the READ MULTI-CHANNEL BUFFER (IO.RBC).

5.8 FORTRAN Interface

The following set of FORTRAN callable subroutines allow FORTRAN programs access to the UDC-11. Handler tasks are normally of a higher priority than tasks requesting handler service, thus there is no delay in reading or writing to the UDC-11. There are implied waits in all subroutines that issue QIO directives. These WAITFOR directives are NOPs except when the requesting task is of a higher priority than the UDC handler task.

5.8.1 ISTS

In the following description, ISTS is a 2-word integer array to receive the results of the call.

Word one of this array always contains a status value that is returned in accordance with the ISA convention as follows:

- ISTS(1) .EQ. 1 - Successful completion
- ISTS(1) .GE. 3 - Request failed.

A failure may occur because either the QIO directive was rejected or the handler detected an error in the request. The following convention is used to distinguish between these conditions.

- For ISTS(1) 300, the Queue I/O directive was rejected and the directive status word DSW = - ISTS(1).
- For ISTS(1) .GT. 300, the request was rejected by the handler task and the handler status word HSW = - (ISTS(1)-300).

The special case of a +3 error return indicates that the subroutine was unable to generate the QIO directive.

ISTS(1) is currently set via an I/O AST internal to the called subroutine. Hence, as a temporary measure, ISTS(1) should not be tested if AST's were disabled when the subroutine was called.

ISTS(2) contains the second word of I/O status returned by the handler.

For a complete description of the UDC FORTRAN calls, the reader is referred to the IAS FORTRAN Special Subroutines Reference Manual.

5.8.2 ASUDLN

FORTRAN call:

```
CALL ASUDLN (LUN,[ISTS])
```

Assigns the specified LUN to UD0 and records it as the logical unit number to be used whenever the logical unit number is unspecified.

5.8.3 AOSC

FORTRAN call:

```
CALL AOSC(ICHN,IVOLTS,[ISTS],[LUN])
```

Sets a given channel to a specified voltage. Voltage is an integer between zero (0v) and 1023 (+10v).

5.8.4 AO/AOW

ISA standard FORTRAN call:

```
CALL AO (INM,ICONT,IDATA,[ISTS],[LUN])
CALL AOW (INM,ICONT,IDATA,[ISTS],[LUN])
```

Performs analog output (without or with a WAIT) on several channels. The number of channels is specified in INM. Channel number and output data are contained in 1-dimensional arrays ICONT and IDATA respectively.

5.8.5 DOSM

FORTRAN call:

```
CALL DOSM(IPT,[ISTS],[LUN])
```

Pulses a single momentary output point. IPT is an integer variable specifying the point number.

5.8.6 DOM/DOMW

FORTRAN call:

```
CALL DOM (INM,ICONT,IDATA,[IDX],[ISTS],[LUN])
CALL DOMW (INM,ICONT,IDATA,[IDX],[ISTS],[LUN])
```

Pulses several 16-point fields. INM specifies the number of fields. ICONT is an integer array containing initial point numbers. The corresponding bit pattern in IDATA specifies which points in a field are to be pulsed.

IDX is a dummy variable retained for compatibility with existing implementations of this call.

5.8.7 DOFM

FORTRAN call:

```
CALL DOFM (IPT,IMSK,[ISTS],[LUN])
```

Pulses one 16-bit string of points. One point is pulsed at each bit position set in IMSK. Bit N corresponds to point IPT+N.

5.8.8 DOSL

FORTRAN call:

```
CALL DOSL (IPT,ISWTCH,[ISTS],[LUN])
```

Activates a latching digital output point (IPT). The point is latched if ISWTCH is .TRUE. (-1) and unlatched otherwise.

5.8.9 DOFL

FORTRAN call:

```
CALL DOFL (IPT,IDATA,IMSK,[ISTS],[LUN])
```

Latches or unlatches a field of 16 points. IPT is an integer variable specifying the initial point in the field. IDATA is an integer variable specifying the points to be latched/unlatched. To change the state of a point, the corresponding bit in IMSK must be set.

5.8.10 DOL/DOLW

FORTRAN call:

```
CALL DOL (INM,ICONT,IDATA,IMSK,[ISTS],[LUN])
CALL DOLW (INM,ICONT,IDATA,IMSK,[ISTS],[LUN])
```

Controls more than one field of latching outputs. The number of fields designated by INM, ICONT, IDATA, and IMSK are single dimension integer arrays that contain the number of entries in the output arrays, initial point number, data, and mask.

5.8.11 RBCD

FORTRAN call:

```
CALL RBCD (IPT,IMSK,ISTS,[LUN])
```

Reads 16 bits of BCD encoded contact sense input under a mask. IPT is an integer variable specifying the initial point in the field. Only those points set in the mask word (IMSK) are read. All other points are input as 0. The result after masking is converted to binary and placed in ISTS(2).

5.8.12 DIFCS

FORTRAN call:

```
CALL DIFCS (IPT,IMSK,ISTS,[LUN])
```

Reads a field of contact sense inputs under a mask. Points not masked are set to zero. The result is in ISTS(2).

5.8.13 DI/DIW

FORTRAN call:

```
CALL DI (INM,ICONT,IDATA,ISTS,[LUN])
CALL DIW (INM,ICONT,IDATA,ISTS,[LUN])
```

Reads several 16-point contact sense fields. The number of fields to be read is specified in INM. The resultant data is placed in IDATA.

ICONT and IDATA are 1-dimensional arrays. ICONT entries specify initial point number. IDATA entries contain the resultant data.

5.8.14 RCSPT

FORTRAN call:

```
CALL RCSPT (IPT,ISTS,[LUN])
```

Reads the state of a single contact sense point into ISTS(2). IPT is an integer specifying the point number to be read. The result is set to .FALSE. (0) if the point is open or .TRUE. (-1) if the point is closed.

5.8.15 RCIPT

FORTRAN call:

```
CALL RCIPT (IPT,ISTS,[LUN])
```

Reads the state of a single contact interrupt point. IPT is an integer specifying the point number to be read. ISTS(2) is set .TRUE. (-1) if the point is closed or .FALSE. (0) if the point is open.

5.8.16 CDTI

FORTRAN call:

```
CALL CDTI (IBUF,ISZ,IEV,[ISTS],[LUN])
```

Connects a circular buffer (IBUF) to receive contact interrupt data. ISZ is the length of the buffer, which must exceed fourteen words. The buffer size required to contain N entries follows:

```
ISZ=(10+5*N)
```

IEV is a trigger event variable to be set whenever the handler attempts to place an entry in the buffer.

5.8.17 RDDI

FORTRAN call:

```
CALL RDDI (IPT,IVAL,[IVRN])
```

Reads the contents of the circular buffer. One point is read for each call. IPT is set < 0 if a valid entry is not found. If the entry is valid, IPT contains the point number and IVAL contains the state. IVRN is an optional integer to receive the overrun count. This count is supplied as a positive nonzero value.

5.8.18 DFDI

FORTRAN call:

```
CALL DFDI ([ISTS],[LUN])
```

Disconnects a buffer.

5.8.19 SCTI

FORTRAN call:

```
CALL SCTI (IMOD,ITM,[ISTS],[LUN])
```

Sets timer module IMOD to an initial value (ITM).

5.8.20 RSTI

FORTRAN call:

```
CALL RSTI (IMOD,ISTS,[LUN])
```

Reads a single timer module (IMOD). The value is placed in ISTS(2).

5.8.21 CTTI

FORTRAN call:

```
CALL CTTI (IBUF,ISZ,IEV,IV,[ISTS],[LUN])
```

Connects a circular buffer (IBUF) to receive timer inputs dynamically. ISZ is the length of the buffer which must exceed eleven. The buffer size required to contain N entries follows:

```
ISZ=(8+4*N)
```

IEV is a trigger event flag to be set whenever the handler attempts to place an entry in the buffer. IV is an array of initial timer values. One entry is required for each timer module in the system.

5.8.22 RDTI

FORTRAN call:

```
CALL RDTI (IMOD,ITM,[IVRN])
```

Reads the contents of the circular buffer. One entry is read for each call. IMOD is set < 0 if the entry is not valid. If the entry is valid, IMOD contains the module number and ITM contains the module number value. IVRN is an optional integer to receive the overrun count. Count is supplied as a positive nonzero value.

5.8.23 DFTI

FORTRAN call:

```
CALL DFTI ([ISTS],[LUN])
```

Disconnects a buffer from timer inputs.

5.8.24 ADU01

For the ADU01, the FORTRAN calls are identical to those for the AFC-11 and AD01. However, when the FORTRAN A/D sample subroutines AIW,AI,AIRD and AISQ are used for ADU01 sampling, the following conditions are required:

1 A LUN must be assigned specifically to the UDC,

2 That LUN must also be specified in the LUN argument of the FORTRAN call.

See the *IAS FORTRAN Special Subroutines Reference Manual.*

5.9 Sample Fortran Program

The following FORTRAN program was written to drive a Demo Panel that implemented the CANCEL and SYNC directives via illuminated switch buttons and ten-position thumbwheel switches.

```
C   TDS -- TASK DISPATCHER TASK FOR SCHEDULE SECTION OF DEMO PANEL.
C
C
C
C
C   MCR FUNCTION: "TDS"
C   FILE NAME: "TDS.N"
C   TASK NAME: "...TDS"
C
C   THE FOLLOWING LATCHING DIGITAL OUTPUT (DOL) AND CONTACT
C   INTERRUPT (CI) POINTS ARE USED TO ILLUMINATE AND DETECT
C   CLOSURES ON THE BUTTONS
C   OF THE SCHEDULE SECTION OF THE PANEL.
C
C   TASK NUMBER SELECTION
C
C   "SELECT" LAMP. DOL #15, SWITCH. CI #15
C
C   CANCEL/ SCHEDULE SELECTION
C
C   "CANCEL" LAMP. DOL #14, SWITCH. CI #14
C   "SCHEDULE" LAMP. DOL #13, SWITCH. CI #13
C
C   SYNCHRONIZATION UNIT SELECTION
C
C   "NOW" LAMP. DOL #12, SWITCH. CI #12
C   "SECOND" LAMP. DOL #11, SWITCH. CI #11
C   "MINUTE" LAMP. DOL #10, SWITCH. CI #10
C   "HOUR" LAMP. DOL #09, SWITCH. CI #09
C
C   PERIODIC RESCHEDULING SELECTION
C
C   "NO" LAMP. DOL #08, SWITCH. CI #08
C   "YES" LAMP. DOL #07, SWITCH. CI #07
C
C   RESCHEDULE UNITS SELECTION
C
C   "TICKS" LAMP. DOL #06, SWITCH. CI #06
C   "SECONDS" LAMP. DOL #05, SWITCH. CI #05
C   "MINUTES" LAMP. DOL #04, SWITCH. CI #04
C   "HOURS" LAMP. DOL #03, SWITCH. CI #03
C
```

```
C   EXECUTE-DISPLAYED-SCHEDULING SELECTION
C
C   "EXECUTE" LAMP. DOL #02, SWITCH. CI #02
C
C   THE TASK NUMBER IS READ FROM THUMBWHEEL
C   DECADES VIA CONTACT SENSE POINTS 00-11
C   (THREE BCD CHARACTERS).
C
C   THE TASK NUMBER IS DISPLAYED ON 7-SEGMENT BCD UNITS
C   WIRED TO THE FOLLOWING LATCHING DIGITAL OUTPUT POINTS:
C
C   32-35 -- ONE'S DIGIT,
C   36-39 -- TEN'S DIGIT.
C   40-43 -- HUNDRED'S DIGIT.
C
C   THE RESCHEDULE INTERVAL MAGNITUDE IS READ FROM THUMBWHEEL DECADES VIA
C   CONTACT SENSE POINTS 16-27 (THREE BCD CHARACTERS).
C
    INTEGER TICKS,SECS,MINS,HOURS
    INTEGER CEFG
    INTEGER TEFG
    INTEGER WEFG
    INTEGER ISTS
    INTEGER DSW
    INTEGER POINT
    INTEGER TSKNUM
    INTEGER TBUF
    INTEGER TSET
    LOGICAL LV
C
    DIMENSION IBUF(40),TASK(10),ISTS(2),TBUF(20),TSET(4)
C
    DATA TICKS,SECS,MINS,HOURS/1,2,3,4/
    DATA CEFG/3/
    DATA TSET(1)/-140/
    DATA TEFG/1/
    DATA WEFG/2/
C
    DATA TASK(01)/RSET/
    DATA TASK(02)/RCHON/
    DATA TASK(03)/RCHREC/
    DATA TASK(04)/RCHOFF/
    DATA TASK(05)/RTIMO/
    DATA TASK(06)/RREGX/
    DATA TASK(07)/RTEMP/
    DATA TASK(08)/RTASK08/
    DATA TASK(09)/RTASK09/
    DATA TASK(10)/RTASK10/
C
C   INITIAL ENTRY -- GET MCR COMMAND LINE (NO PARAMETERS ARE
C   TAKEN, THIS CALL JUST FREES THE MCR COMMAND LINE BUFFER)
C
    CALL GETMCR (IBUF)
C
C   ASSIGN AND RECORD LUN-10 AS UDC
C
    CALL ASUDLN (10)
C
C   CONNECT CIRCULAR BUFFER "IBUF" TO RECEIVE CONTACT INTERRUPT INFO.
C   IF FAILURE TO CONNECT (OTHER TASK CONNECTED). "STOP 1".
C
    CALL CTDI (IBUF,36,TEFG,ISTS)
    IF (ISTS(1) .GE. 3) STOP 1
```

```
C
C
C   CONNECT CIRCULAR BUFFER "TBUF" TO RECEIVE TIMER INFO.
C   IF FAILURE TO CONNECT (OTHER TASK CONNECTED). "STOP 2".
C
    CALL CTTI (TBUF,20,WEFG,TSET,ISTS)
    IF (ISTS(1).GE. 3) STOP 2
C
C
C   100 -- START OF SCHEDULING SEQUENCE.
C
C   (1) TURN OFF 7-SEGMENT (LED) TASK NUMBER DISPLAY, BY SETTING
C     BCD DIGITS FIFTEEN
C   (2) TURN OFF ALL BUTTON LAMPS ON SCHEDULE SECTION
C     OF PANEL (DOL POINTS 2-25)
C   (3) FLASH "SELECT" BUTTON LAMP (DOL #15) UNTIL BUTTON IS
C     PRESSED (CONTACT CLOSURE ON CI #15)
C
100  CALL DOL (1,32,15,"17)
    CALL DOL (1,36,15,"17)
    CALL DOL (1,40,15,"17)
C
C   110 -- RE-START AFTER SUCCESSFUL SCHEDULE OR CANCEL -- LEAVE
C       TASK NUMBER DISPLAYED IN 7-SEG LED'S.
C
110  DO 112 J=2,15
112  CALL DOSL (J,.FALSE.)
    CALL CLREF (TEFG)
    CALL CLREF (WEFG)
    ITGL=.TRUE.
C
115  CALL DOSL (15,ITGL)
    ITGL=IEOR(.TRUE.,ITGL)
116  CALL WFLOR (TEFG,WEFG)
    CALL READEF(TEFG,DSW)
    IF (DSW.EQ.2) GO TO 132
117  CALL RDTI(NTM,ITIM)
    IF(NTM)116,115,117
C
C   CHECK FOR CONTACT CLOSURE ON "SELECT" BUTTON.
C
C
132  CALL RDDI (POINT,LV)
    IF (POINT .LT. 0) GO TO 110
    IF (POINT .NE. 15) GO TO 132
    IF (LV .EQ. .FALSE.) GO TO 132
C
C   140 -- A CONTACT CLOSURE HAS BEEN DETECTED ON THE "SELECT" SWITCH.
C   WAIT FOR CONTACT BOUNCE TO STOP ("SELECT" BUTTON IS USED AS A
C   "RESET" KEY THROUGHOUT SCHEDULING SEQUENCE, AND THEREFORE,
C   CONTACT BOUNCE IS UNDESIRABLE).
140  CALL RDDI (POINT,LV)
    IF (POINT .GE. 0) GO TO 140
    CALL MARK (CEFG,5,TICKS)
    CALL WFLOR (TEFG,CEFG)
    CALL READEF (TEFG,DSW)
    IF (DSW .EQ. 2) GO TO 140
C
C   A TASK NUMBER HAS BEEN SELECTED (VIA "SELECT" BUTTON AND THUMBWHEEL
C   DECADES), TURN "SELECT" BUTTON LAMP ON, AND USE "SELECT" BUTTON
C   AS A "RESET" KEY. I.E., IF PRESSED DURING SCHEDULE SELECTION
C   SEQUENCE; THE SEQUENCE IS RESTARTED (AT STATEMENT #100).
C
```

```
      CALL DOSL (15,.TRUE.)
C
C     READ TASK NUMBER FROM THUMBWHEEL DECADE SWITCHES & DISPLAY
C     TASK NUMBER IN 7-SEGMENT LED DISPLAY UNITS.
C
      CALL RBCD (00,"007777,ISTS)
      TSKNUM=ISTS(2)
C
C     "TSKNUM" CONTAINS THE TASK NUMBER THRUOUT SCHEDULING SEQUENCE
C
      NUM=ISTS(2)
      N=NUM/100
      CALL DOL (1,40,N,"17)
      NUM=NUM-100*N
      N=NUM/10
      CALL DOL (1,36,N,"17)
      NUM=NUM-10*N
      CALL DOL (1,32,NUM,"17)
C
C     SPECIAL CASE: IF TASK NUMBER 000, EXIT TASK DISPATCHER
C
      IF (TSKNUM .EQ. 000) GO TO 900
C
C     SELECT TASK SCHEDULING OR CANCELING BY:
C
C     (1) TURNING ON BOTH THE "CANCEL" & "SCHEDULE" BUTTON LAMPS AND
C     (2) WAITING FOR A CONTACT CLOSURE FROM EITHER "CANCEL", "SCHEDULE",
C       OR "SELECT" BUTTON SWITCHES.
C
      CALL DOSL (13,.TRUE.)
      CALL DOSL (14,.TRUE.)
C
202   CALL WAITFR (TEFG)
204   CALL RDDI (POINT,LV)
      IF (POINT .LT. 0) GO TO 202
      IF (LV .EQ. .FALSE.) GO TO 204
      IF (POINT .EQ. 13) GO TO 250
      IF (POINT .EQ. 14) GO TO 240
      IF (POINT .EQ. 15) GO TO 100
      GO TO 204
C
C     240 -- A CONTACT CLOSURE HAS BEEN DETECTED ON THE "CANCEL"
C     BUTTON SWITCH
C
C     (1) TURN OFF THE "SCHEDULE" BUTTON LAMP
C     (2) TURN ON THE "EXECUTE" BUTTON LAMP
C     (3) WAIT FOR A CONTACT CLOSURE ON EITHER "EXECUTE" OR "SELECT".
C
240   CALL DOSL (13,.FALSE.)
      CALL DOSL (02,.TRUE.)
C
242   CALL WAITFR (TEFG)
244   CALL RDDI (POINT,LV)
      IF (POINT .LT. 0) GO TO 242
      IF (LV .EQ. .FALSE.) GO TO 244
      IF (POINT .EQ. 15) GO TO 100
      IF (POINT .NE. 02) GO TO 244
C
C     A CONTACT CLOSURE HAS BEEN DETECTED ON THE "EXECUTE" BUTTON SWITCH.
C
C     (1) TURN OFF "EXECUTE" BUTTON LAMP
C     (2) CANCEL TASK PER "TSKNUM"
C     (3) RESTART SCHEDULING SEQUENCE.
```

```
C
      CALL DOSL (2,.FALSE.)
      CALL CANALL (TASK(TSKNUM),DSW)
      IF (DSW .GT. 0) GO TO 110
      GO TO 100
C
C     250 -- A CONTACT CLOSURE HAS BEEN DETECTED ON THE "SCHEDULE"
C     BUTTON SWITCH. SELECT SYNCHRONIZATION UNITS AS FOLLOWS.
C
C     (1) TURN OFF THE "CANCEL" BUTTON LAMP
C     (2) TURN ON "HOUR", "MINUTE", "SECOND", & "NOW" BUTTON LAMPS
C     (3) WAIT FOR A CONTACT CLOSURE ON EITHER "HOUR", "MINUTE", "SECOND",
C     "NOT", OR "SELECT" BUTTON SWITCHES.
C
250   CALL DOSL (14,.FALSE.)
      CALL DOSL (09,.TRUE.)
      CALL DOSL (10,.TRUE.)
      CALL DOSL (11,.TRUE.)
      CALL DOSL (12,.TRUE.)
C
252   CALL WAITFR (TEFG)
254   CALL RDDI (POINT,LV)
      IF (POINT .LT. 09) GO TO 252
      IF (LV .EQ. .FALSE.) GO TO 254
      IF (POINT .EQ. 15) GO TO 100
      IF (POINT .LT. 09) GO TO 254
      IF (POINT .GT. 12) GO TO 254
C
C     A CONTACT CLOSURE HAS BEEN DETECTED ON A POINT BETWEEN #9 AND
C     #12 ("HOUR", "MINUTE", "SECOND", OR "NOW" BUTTON SWITCHES).
C     CONVERT POINT NUMBER TO SYNC UNITS ("ISYU"), AND TURN OFF
C     BUTTON LAMPS FOR SYNC UNITS NOT SELECTED.
C
      ISYU=13-POINT
C
      DO 256 J=9,12
      IF (J .EQ. POINT) GO TO 256
      CALL DOSL (J,.FALSE.)
256   CONTINUE
C
C     SELECT PERIODIC RE-SCHEDULING BY:
C
C     (1) TURNING ON BOTH "YES" & "NO" BUTTON LAMPS, AND
C     (2) WAITING FOR A CONTACT CLOSURE ON EITHER "YES", "NO", OR
C       "SELECT".
C
      CALL DOSL (7,.TRUE.)
      CALL DOSL (8,.TRUE.)
C
262   CALL WAITFR (TEFG)
264   CALL RDDI (POINT,LV)
      IF (POINT .LT. 0) GO TO 262
      IF (LV .EQ. .FALSE.) GO TO 264
      IF (POINT .EQ. 07) GO TO 270
      IF (POINT .EQ. 08) GO TO 268
      IF (POINT .EQ. 15) GO TO 100
      GO TO 264
C
C     268 -- A CONTACT CLOSURE HAS BEEN DETECTED ON THE "NO"
C     BUTTON SWITCH.
C
C     (1) TURN OFF "YES" BUTTON LAMP
C     (2) INDICATE NO RE-SCHEDULING ("IRI"="IRU"=0)
```

```
C   (3) ILLUMINATE AND WAIT FOR "EXECUTE"
C
268  CALL DOSL (7,.FALSE.)
C
    IRI=0
    IRU=0
C
    GO TO 300
C
C   270 -- A CONTACT CLOSURE HAS BEEN DETECTED ON THE "YES" BUTTON
C   SWITCH. SELECT RE-SCHEDULING INTERVAL UNITS BY:
C
C   (1) TURN OFF "NO" BUTTON LAMP
C   (2) TURN ON "HOURS", "MINUTES", "SECONDS", & "TICKS" BUTTON LAMPS
C   (3) WAIT FOR A CONTACT CLOSURE ON EITHER "HOURS", "MINUTES",
C   "SECONDS", OR "SELECT" BUTTON SWITCHES
C
270  CALL DOSL (8,.FALSE.)
    CALL DOSL (3,.TRUE.)
    CALL DOSL (4,.TRUE.)
    CALL DOSL (5,.TRUE.)
    CALL DOSL (6,.TRUE.)
C
282  CALL WAITFR (TEFG)
284  CALL RDDI (POINT,LV)
    IF (POINT .LT. 0) GO TO 282
    IF (LV .EQ. .FALSE.) GO TO 284
    IF (POINT .EQ. 15) GO TO 100
    IF (POINT .LT. 3) GO TO 284
    IF (POINT .GT. 6) GO TO 284
C
C   A CONTACT CLOSURE HAS BEEN DETECTED ON A CONTACT INTERRUPT POINT
C   BETWEEN #3 AND #6 ("HOURS", "MINUTES", "SECONDS", OR "TICKS").
C
C   (1) TURN OFF BUTTON LAMPS FOR RE-SCHEDULE UNITS NOT SELECTED
C   (2) CONVERT POINT NUMBER TO TIME UNIT INDICATOR
C   (3) READ RE-SCHEDULE INTERVAL MAGNITUDE FROM THUMBWHEEL SWITCHES
C
    DO 286 J=3,6
    IF (J .EQ. POINT) GO TO 286
    CALL DOSL (J,.FALSE.)
286  CONTINUE
C
    IRU=7-POINT
C
    CALL RBCD (16,"007777,ISTS)
    IRI=ISTS(2)
C
C   300 -- TASK SCHEDULING PARAMETERS ARE DISPLAYED ON PANEL.
C   PERFORM OR REJECT BY:
C
C   (1) TURNING ON "EXECUTE" LAMP, AND
C   (2) WAITING FOR A CONTACT CLOSURE ON EITHER "EXECUTE" OR "SELECT".
C
300  CALL DOSL (02,.TRUE.)
C
302  CALL WAITFR (TEFG)
304  CALL RDDI (POINT.LV)
    IF (POINT .LT. 0) GO TO 302
    IF (LV .EQ. .FALSE.) GO TO 304
    IF (POINT .EQ. 15) GO TO 100
    IF (POINT .NE. 02) GO TO 304
C
```

```
C    A CONTACT CLOSURE HAS BEEN DETECTED ON "EXECUTE" BUTTON
C    SWITCH, SYNC TASK.
C
     IF (TSKNUM .LT. 1) GO TO 100
     IF (TSKNUM .GT. 10) GO TO 100
     CALL SYNC (TASK(TSKNUM),9,SECS,ISYU,IRI,IRU,DSW)
     IF (DSW .GT. 0) GO TO 110
     GO TO 100
C
C    900 -- SPECIAL CASE: TASK #000 -- EXIT DISPATCHER.
C
900   CALL DFDI
     CALL DFTI
C
     DO 910 J=2,15
910   CALL DOSL (J,.TRUE.)
C
     CALL EXIT
C
     END
```

5.10 UDC STATUS RETURNS

IOST contains a code indicating the disposition of the QIO request. These status return codes for the UDC-11 handler are symbolized as shown below.

Symbol	Meaning
IS.SUC	Successful completion
IE.BAD	Bad parameters
IE.PRI	Privilege violation
IE.MOD	Invalid UDC module
IE.CON	UDC connect error
IE.SPC	Part of buffer is out of address space
IE.IFC	Invalid function code

See Appendix A for a complete list of I/O status returns.

6 DECtape Handler

6.1 DECtape Handler Functions

The TC-11/TU56 DECtape system is controlled by the DECtape handler task. The handler task supports the TC-11 DECtape controller and up to 4 TU56 DECtape units (that is, 8 DECtape drives). The handler is a single controller handler, but multiple copies can service additional TC-11 controllers for systems with more than 8 DECtape drives.

The DECtape handler is installed with DT.... as the task name.

6.2 Function Codes

I/O requests serviced by the DECtape handler are issued using the QIO$ system macro. See Section 1.7 for a detailed discussion of function codes for mass storage devices. The QIO$ macro calls follow.

6.2.1 READ/WRITE Logical Functions

```
QIO$ fc,lun,ef,pri,iosb,ast,<stadd,size,wd3,wd4,lbn>
```

fc can have one of the following values.

- IO.RLB - Read logical block (forward)
- IO.RLV - Read logical block (reverse)
- IO.WLB - Write logical block (forward)
- IO.WLV - Write logical block (reverse)

The five parameter words bracketed by left and right angle brackets (<>) must be specified and must be delimited by the angle brackets.

They have the following meaning:

- stadd - Address of I/O buffer in user's virtual space (this value must be even).
- size - Length of transfer in bytes (this value must be even and nonzero).
- wd3 - Ignored (this value must be represented by a zero).
- wd4 - Ignored (this value must be represented by a zero).
- lbn - Logical block number (0-577. inclusive).

6.2.2 ATTACH, DETACH, and REWIND Functions

```
QIO$ fc,lun,ef,pri,iosb,ast
```

fc can have one of the following values.

- IO.RWD - Rewind DECtape unit
- IO.RWU - Rewind and unload DECtape unit
- IO.ATT - Attach DECtape unit
- IO.DET - Detach DECtape unit

6.2.3 DECtape Transfers

PDP-11 DECtapes are divided into 256-word blocks. If, on a WRITE, the transfer length is less than 256 words, a partial block is transferred with zero fill for the rest of the physical block. If, on a READ, the transfer length is less than 256 words, only the number of words specified are transferred. If the transfer length is greater than 256 words, more than one physical block is transferred.

6.2.4 DECtape READ/WRITE

The DECtape handler supports READ/WRITE in reverse direction as well as forward. Normally a block should be read in the same direction as it was written. If a block is read in the opposite direction from which it was written, it is reversed in memory; that is, word 255 becomes word 0 and word 254 becomes word 1.

6.3 UNIBUS Mapping Registers

All DMA devices on the UNIBUS of a PDP-11/44 or a PDP-11/70 use UNIBUS mapping registers (UMRs) to perform DMA transfers if the machine is running in 22-bit mode (See the appropriate PDP-11 Processor Handbook). The DECtape handler attempts to allocate only one UMR because all transfers are buffered in the handler. If the handler cannot allocate a UMR on initialization, it exits.

6.4 Error Handling

The DECtape handler performs special handling for the select error condition. If a select error occurs during the execution of a READ, WRITE, or REWIND, the message

*** SELECT ERROR ON DTn
n = unit number

is printed on the command output (CO) device (logical unit 2 for the driver). The DECtape handler then does not dequeue normal user requests for that unit until the error is remedied by the operator. The only user request it will dequeue for that unit is Cancel (see Section 1.6.4). Other errors simply return a negative value in the low byte of the I/O status block.

The specific errors that can be returned by the DECtape handler are listed below.

6.5 DT Status Returns

IOST contains a code indicating the disposition of the QIO request. These status return codes for the DECtape handler are symbolized as shown below.

Symbol	Meaning
IE.BAD	Bad parameters.
IE.IFC	Invalid function code.
IE.DNR	Select error (only occurs if the handler fails to send its select error messages).
IE.VER	Fatal error in READ or WRITE (other than Mark Track Error). IOST+2 contains DECtape error flags. This is normally a parity error; however, the error may also be caused by performing a multi-block transfer past block 1101 (577.), or in reverse direction past block zero, in which case the end zone (ENDZ) error flag is set in IOST+2. The operation is tried 5 times by the handler before an error is reported to the user. See the *PDP-11 Pheripherals Handbook.*
IE.SPC	Part of the user's buffer is out of user's virtual space, a byte count of zero was specified, or on the PDP-11/70, insufficient UMRs are available to handle the transfer.
IE.DNA	Detach failed.
IE.DAA	Attach failed (device already attached).
IE.WLK	Write lock error
IE.SRE	SEND/REQUEST failure when passing information to FILES-11 interfaces.
IE.ABO	Operation aborted (while in Mark Time Wait). Either a handler exit or I/O rundown forced operation to abort.
IE.PRI	Access privilege violation.
IE.BYT	Odd transfer address or byte count.
IE.BLK	Logical block number greater than 1101 octal (577 decimal).
IE.BBE	Bad block error (mark track error on read or write). IOST+2 contains the number of bytes transferred at the point of the error so the actual bad block number may be determined. The operation is tried 5 times by the handler before an error is reported to the user.

6.6 Characteristics Words for DECtape

See Section 1.8 for the four characteristics words set or implied at System Generation and stored in the system's PUD entry for each individual unit. DECtape is a random-access device so that words 2 and 3 are laid out in the same way as for disks (See Section 4.6). DECtape is analogous to a disk with one block per cylinder and having to seek in order to access any block. The settings for words 2 and 3 are as follows:

word 2 (offset U.C2 from the PUD entry):

bit 0	U2.WCK	Ignored by the DECtape handler
bits 1-3		reserved
bit 4	U2.MOH	set
bit 5	U2.RMV	set
bit 6	U2.BAD	clear
bit 7		reserved

bits 8-12	10 (octal)
bits 13-15	reserved

word 3 (offset U.C3 from the PUD entry):

low byte	1
high byte	1

The normal setting for words 2 and 3 is thus 4060,401 for a DECtape unit.

7 Magnetic Tape Handlers

7.1 Magtape Handler Functions

The magtape handlers provide the user with access to the TU10/16, TE10/16, TU77, and TU45 industry-compatible magnetic tape units. Table 7–1 relates the devices to the handler tasks that reference them.

Table 7–1 Standard Magnetic Tape Devices

Device Driver	Installed Task Name	Channels	Recording Density (Frames/ Inch)	Tape Speed (Inches/ Second)	Maximum Data Rate Units	Recording Transfer Method (Bytes/ Second)
TE10 TU10	MT....	9 7 or 9	7-channel: 200, 556 or 800 9-channel: 800	45	36,000	NRZI
TE16,TU16	MM....	9	800/1600	45	800 bpi: 36,000 1600 bpi: 72,000	NRZI or PE [1]
TU45	MM....	9	800/1600	75	800 bpi: 60,000 1600 bpi: 120,000	NRZI or PE [1]

[1] Phase encoded
[2] Low speed
[3] High speed
[4] Serial serpentine
[5] In streaming mode

Table 7–1 (Cont.) Standard Magnetic Tape Devices

Device Driver	Installed Task Name	Channels	Recording Density (Frames/ Inch)	Tape Speed (Inches/ Second)	Maximum Data Rate Units	Recording Transfer Method (Bytes/ Second)
TU77	MM....	9	800/1600	125	800 bpi: 100,000 1600 bpi: 200,000	NRZI or PE [1]
TS03	MT....	9	800	15	12,000	NRZI
TS11	MS....	9	1600	45	72,000	PE[1]
TU80	MS....	9	1600	25[2] 100[3]	40,000[2] 160,000[3]	PE[1]
TU81	MU....	9	1600/6250	25[2] 75[3] 25[2] 75[3]	40,000 120,000 156,000 469,000	PE[1] PE[1] GCR GCR
TS05	MS....	9	1600	25	40,000	PE[1]
TK25	MS....	s.s.[4]	8000	55	55,000 bit-serial data tracks recorded serial serpentine	Modified GCR

[1] Phase encoded
[2] Low speed
[3] High speed
[4] Serial serpentine
[5] In streaming mode

Table 7–1 (Cont.) Standard Magnetic Tape Devices

Device Driver	Installed Task Name	Channels	Recording Density (Frames/ Inch)	Tape Speed (Inches/ Second)	Maximum Data Rate Units	Recording Transfer Method (Bytes/ Second)
TK50	MU....	s.s.[4]	6667	75[5]	45,000 bit-serial data tracks recorded serial serpentine	Modified FM

[1] Phase encoded
[2] Low speed
[3] High speed
[4] Serial serpentine
[5] In streaming mode

The requesting task can perform reads, writes, and positioning operations at a selectable density and parity setting. The handlers also provide error recovery facilities that will automatically retry tape operations a number of times before reporting error status. The handlers can execute any of the following functions:

1 Read logical record

2 Write logical record

3 Attach unit

4 Detach unit

5 Device Control Functions

   a. Rewind magtape

   b. Skip n records (forward or reverse)

   c. Skip n files (forward or reverse)

   d. Set tape characteristics (parity/density, etc.)

   e. Read tape characteristics

   f. Rewind and turn unit off line

   g. Verify tape is at load point and set characteristics

6 Write End-of-File character

   - An End-of-File character (EOF) is a special mark used to separate data sets. ANSI uses the equivalent term "Tape Mark".

When a request fails because the desired unit is off line, the magtape handler prints:

***MAGTAPE SELECT ERROR ON MTn
n = unit number

on the operator console. Operations on other units are allowed to proceed while one unit is held up due to this select error condition.

7.1.1 TE10/TU10/TS03 Magnetic Tape

The TE10/TU10/TS03 consists of a TM11 controller with a TE10, 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) format.

7.1.2 TE16/TU16/TU45/TU77 Magnetic Tape

The TE16/TU16/TU45/TU77 consists of an RH11/RH70 controller, a TM02 or TM03 formatter, and a TE16/TU16/TU45/TU77 transport. They are quite similar to the TE10/TU10 but are MASSBUS devices, with a common controller, a specialized formatter, and drives. Recording is either 800 bits per inch (bpi) NRZI or 1600 bpi phase encoded (PE).

7.1.3 TS11/TU80 Magnetic Tape

The TS11 and TU80 are integrated subsystems. Each has a drive, a controller, and a formatter. The hardware is microprocessor controlled for all operations, including I/O transfers and tape motion, and it has comprehensive (internal) diagnostic test execution. Recording is 1600 bpi PE.

The TS11 operates in conventional start and stop mode while the TU80 operates at either low speed (start and stop mode) or high speed (streaming mode). Tape speed is microprocessor controlled.

7.1.4 TS05 Magnetic Tape

The TS05 tape subsystem runs on UNIBUS or Q-bus subsystems. It is an intergrated subsystem with a drive, a controller, and a formatter. The hardware is microprocessor controlled for all operations, including I/O transfers tape motion, and it has comprehensive (internal) diagnostic test execution. Recording is 1600 bpi PE. The TS05 operates at 25 inches per second.

7.1.5 TK25 Magnetic Tape

The TK25 consists of a TKQ25 controller for the Q-bus and a TK25 streaming tape drive. The integrated subsystem consists of a tape drive and controller/ formatter. The TK25 uses a DC600A 1/4-inch tape cartridge and stores data on serial data tracks in a serial serpentine recording method. The TK25 has storage capacity of 60 megabytes (Mb) for 8-kilobyte (Kb) data records. Data recording is an 8000 bpi, modified GCR (group cyclical recording) method.

7.1.6 TK50 Magnetic Tape

The TK50 is an integrated subsystem that consists of a controller for the Q-bus (TQK50) or a controller for the UNIBUS (TUK50), and a TK50 streaming tape drive. The controller handles all error recovery and correction, and internally buffers multiple outstanding commands. The tape drive and writes data on 1 1/2-inch tape cartridge that is records at 6667 bpi on serial data tracks in a serial serpentine recording (Modified Frequency Modulation) method. The tape speedis 75 inches per second in streaming mode and the storage capacity is approximately 94 Mb irrespective of record size. There is one drive for each controller.

7.1.7 TU81 Magnetic Tape

The TU81 is a 9-track streaming tape drive that reads and writes data at either 6250 bpi (GCR) or 1600 bpi (PE) on 1/2-inch tape. The TU81 internally buffers multiple outstanding commands. The tape transport speed is 25 or 75 inches per second and is microprocessor controlled. At 6250-bpi density, the drive can store up to 140 Mb on a standard 2400-foot reel. The TU81 has its own UNIBUS controller (one drive per controller).

7.2 Function Codes

I/O requests serviced by the magtape handler are issued using the QIO$ system macro. See Section 1.7 for a detailed discussion of function codes for mass storage devices. The QIO$ macro calls follow.

7.2.1 READ/WRITE Logical Functions

```
QIO$ fc,lun,ef,pri,iosb,ast,<stadd,size>
```

fc can have one of the following values:

- IO.RLB - Read logical block (see Section 7.2.3)
- IO.WLB - Write logical block (see Section 7.2.4)

The two parameter words stadd and size, must be specified and delimited by the angle brackets (<>). Parameters follow:

- stadd - Address of I/O buffer in user's virtual space (this value must be even)
- size - Length of transfer in bytes (this value must be even and non-zero)

7.2.2 ATTACH, DETACH, REWIND, and EOF Functions

```
QIO$  fc,lun,ef,pri,iosb,ast
```

fc can have one of the following values:

- IO.ATT - Attach magtape unit
- IO.DET - Detach magtape unit
- IO.RWD - Rewind magtape unit (see Section 7.2.6)
- IO.RWU - Rewind and turn unit offline (see Section 7.2.5)
- IO.EOF - Write an end-of-file (EOF) character on the tape to mark the end of a data file.

For Skip, Set Characteristics and Verify Functions see Section 7.3.

7.2.3 Read Logical Block

The read function causes the next record on the magtape to be read into the requesting task's input buffer. On completion, IOST+2 contains the length, in bytes, of the record that was read.

Note that read returns an error, IE.DAO, if the physical record size exceeds the specified byte count for the transfer. If this occurs, the first n bytes (where n is the specified byte count) are actually transferred into memory and the remainder of the record is checked for parity but not transferred. If the physical record size is less than the size of the specified byte count, only data for that record is transferred. The byte count is in IOST+2 and a success condition is returned.

7.2.4 Write Logical Block

The write function causes the contents of the I/O buffer to be written as a single physical record on the magtape. Note the restriction that the record size (that is, buffer length) must be at least 14 bytes. The maximum record size is 65535 bytes; however, it is not suggested that such large records be used. A more reasonable upper limit would be 2K bytes.

If the handler detects a parity error when writing a record, the handler backspaces and retries the write automatically. If the error persists after five retries, the handler attempts to write with extended interrecord gap. This enables the record to be placed three inches farther down the tape, past the (presumed) bad spot on the tape. The write with extended interrecord gap operation is also attempted five times before an error is reported to the requesting task. If, for some reason, the requesting task wishes to prohibit write with extended interrecord gap from occurring, it may do so by utilizing the set characteristics functions. (See Section 7.3)

7.2.5 Rewind and Turn Unit Off Line

This command ensures that the unit is turned off line. It is normally used when operator intervention is necessary (for example, when loading a new tape is required). The operator will have to turn the unit manually on line before subsequent operations proceed.

7.2.6 Rewind Magnetic Tape Unit

This command causes the magnetic tape unit to rewind. When the rewind is initiated, the handler immediately issues an I/O done status, (IS.SUC) to the user task.

The immediate return of I/O done allows the user task to continue processing without having to wait for the rewind to complete.

Additional QIO functions issued to the unit being rewound will not execute until the rewind is completed.

7.3 Device Control Function Codes

The Skip, Set Characteristics and Verify function codes are described in the following separate paragraphs with figures and charts for clarity.

7.3.1 Skip n Records

For this function I/O requests serviced by the magtape handler are issued by the QIO$ macro with the following format:

```
QIO$   IO.SPB,lun,ef,pri,iosb,ast,<nrs>
```

The parameter word must be specified and enclosed by left and right angle brackets (<>). It has the following meaning:

- nrs - Number of records to skip

The skip-records function causes the tape unit to skip forward or reverse over a number of physical blocks on the tape. If nrs is greater than zero, it is taken as the number of records to skip in the forward direction; if nrs is less than zero, then the tape is backspaced nrs records. See Section 7.3.6 for end of volume considerations. If nrs equals zero, the handler task returns a status of IS.SUC.

IOSB+2 contains the actual number of records skipped (counting the EOF character as one record). Note that attempting to backspace over the Beginning-of-Tape (BOT) is not considered an error; however, backspacing stops at load point on encountering BOT and the actual number of records skipped is returned in IOST+2.

7.3.2 Skip n Files

For this function I/O requests serviced by the magtape handler are issued by the QIO$ macro with the following format:

```
QIO$   IO.SPF,lun,ef,pri,iost,ast,<ncs>
```

The parameter word must be specified and enclosed with left and right angle brackets (<>). It has the following meaning:

- ncs - Number of EOF characters to skip

The skip files function causes the tape unit to skip forward or reverse until encountering the specified number of EOF characters. If ncs is greater than zero, it specifies the number of EOF characters to skip in the forward direction; if ncs is less than zero, the tape is backspaced over ncs EOF characters. See Section 7.3.6. for end of volume considerations. If ncs is zero, success is returned in IOST.

7.3.3 Set Characteristics

For this function, I/O requests serviced by the magtape handler are issued by the QIO$ macro. The macro has the following format:

```
QIO$    IO.STC,lun,ef,pri,iosb,ast,<cb>
```

The parameter word must be specified and enclosed with left and right angle brackets (<>). It has the following meaning:

- cb - characteristics bits to set

This function allows a task to set certain characteristics bits. These bits are defined in Figure 7–1.

A task which uses magtape should always set the tape characteristics to the proper value since it cannot be certain what state they were left in by the previous task. (See Figure 7–2 and Figure 7–3.)

7.3.4 Read Characteristics

For this function, I/O requests serviced by the magtape handler are issued by the QIO$ macro. The macro has the following format:

```
QIO$    IO.SEC,lun,ef,pri,iosb,ast
```

This function returns the tape characteristics word in IOST+2.

Note that this function always succeeds and never causes the MAGTAPE SELECT ERROR message to be issued. The fact that the unit is off-line (select error) or is rewinding is reported in the bits defined in Figure 7–1.

7.3.5 Verify Beginning of Tape and Set Characteristics

The I/O requests serviced by the magtape handler to verify that the tape is at load point and to set its characteristics are issued by the QIO$ macro in the following format:

```
QIO$   IO.SMO,lun,ef,pri,iost,ast,<cb>
```

The parameter word must be specified and enclosed with left and right angle brackets (<>). It has the following meaning:

- cb - Characteristics bit to set

This function first selects the unit to ensure that it is on line and positioned at load point and then sets the characteristics bits. See Section 7.1 for a description of the tape characteristics bits.

If the tape is not at load point, an error (IE.FHE) is returned to the requesting task (the characteristics bits are not set).

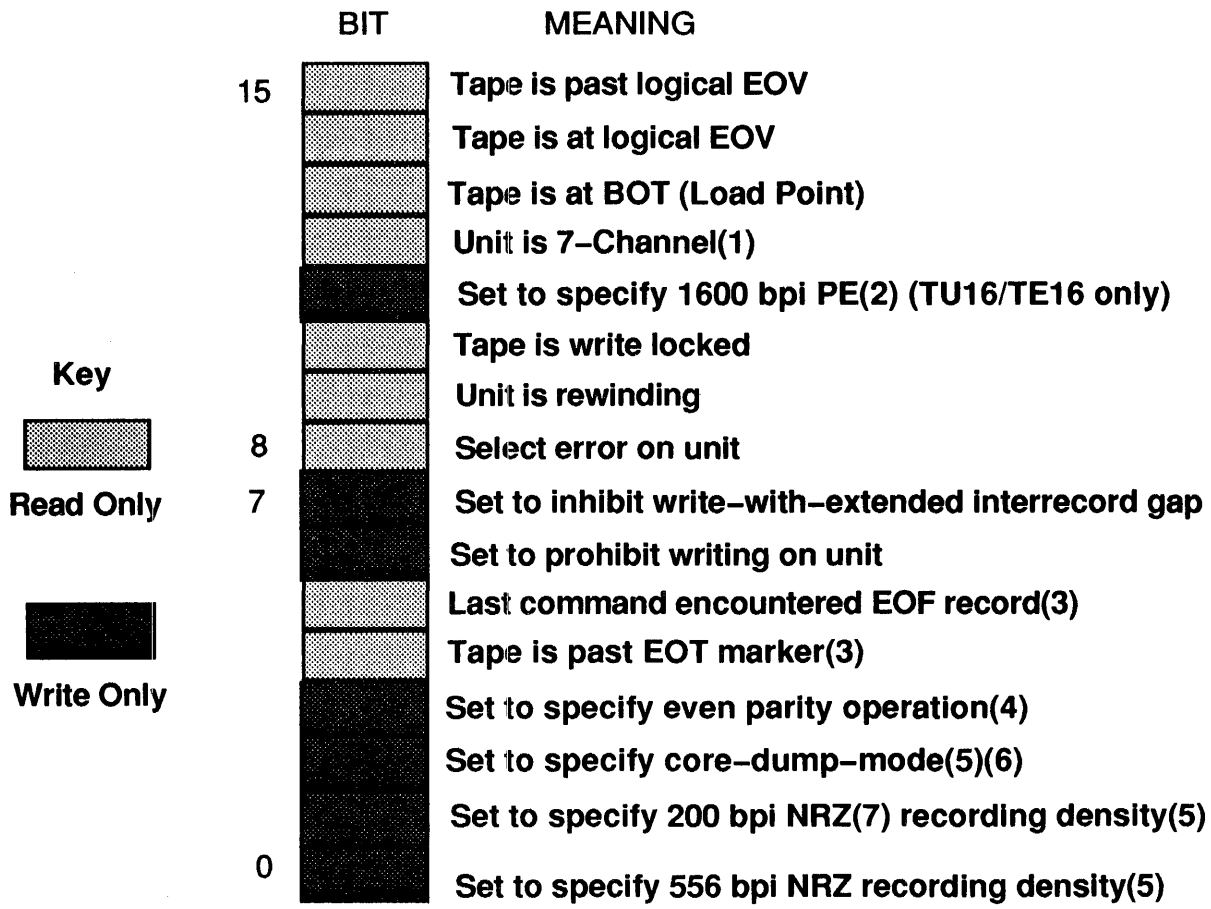
7.3.6 Logical End-of-Volume (EOV)

EOV is defined as two EOF characters in immediate succession. See Figure 7–4 for an illustrated description of how this works.

The EOV state applies to the IO.SPB (Section 7.3.1) and IO.SPF (Section 7.3.2) functions only. The IE.EOV status return can be used to locate the logical end-of-volume so that a new file can be added to the tape. In this case, the requesting task executes an IO.SPF where parameter ncs is greater than the actual number of files on the tape. When logical end-of-volume is detected, an IO.EOV is returned in IOST and IOST+2 contains a count of the actual number of files skipped.

In special case B Figure 7–4, use the Read characteristics function, IO.SEC, to determine if the tape is situated at logical end-of-volume or actually at physical end-of-volume (EOT).

Figure 7–1 Set/Sense Characteristics Status Word



(1) TU16 available in 9–channel only.
(2) Phase encoded.
(3) Cleared by set characteristics.
(4) A unit with even parity set cannot write characters of all zeros so the null set is translated to 020.
(5) 7–channel drives only. See Figure 7–2. The default status is 000004, core dump mode. This is the initial setting when the driver is loaded.
(6) For 7–track units, the use of normal mode results in the loss of the upper 2 bits of each byte.
(7) Non–return–to–zero.

7.4 MT Status Returns

IOST contains a code indicating the disposition of the QIO request. These status return codes for the magtape handler are symbolized as shown below.

Figure 7–2 TU10 Parity/Density Determination

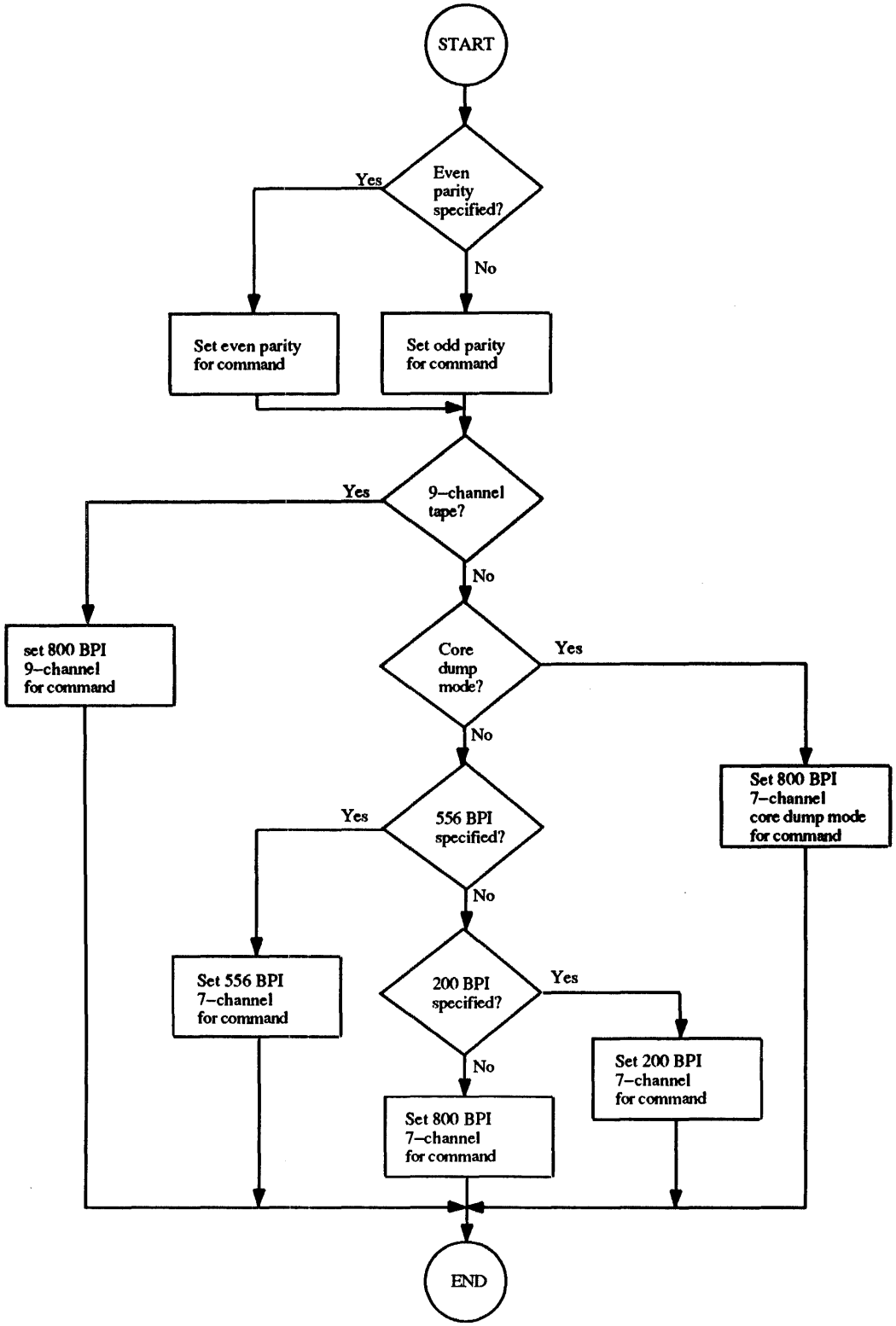


Figure 7–3 TU16 Parity/Density Determination

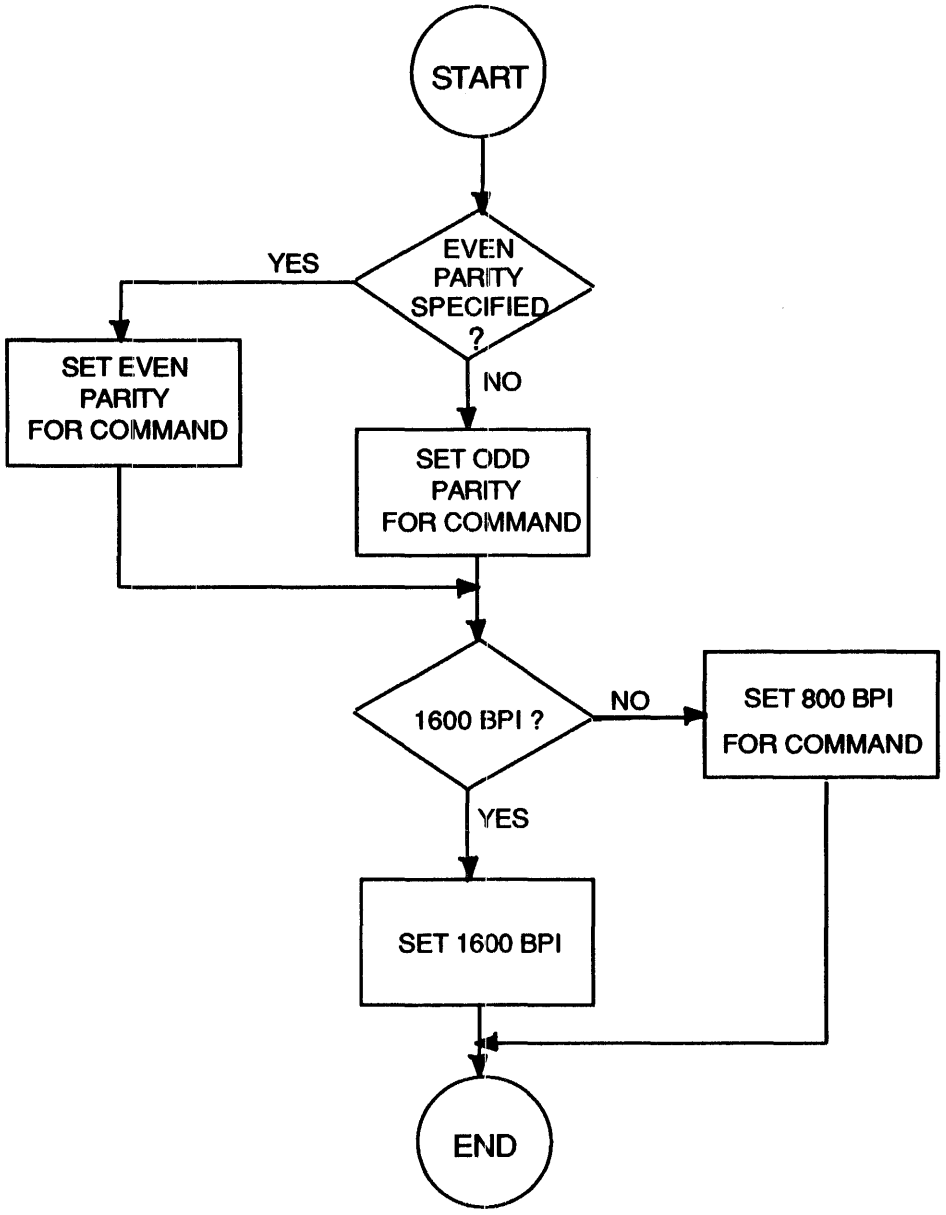
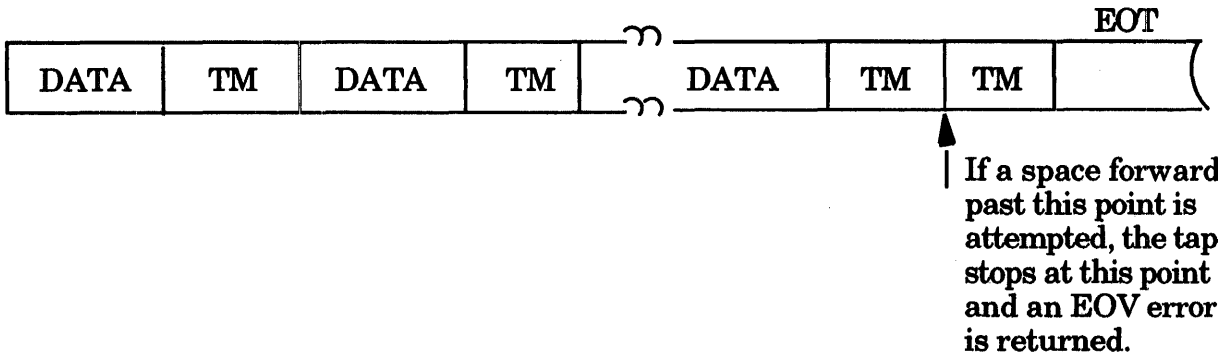


Figure 7–4 Logical End of Volume (EOV)



Key: TM Indicates tape mark, which is an EOF character written on the tape. This is sometimes referred to as an EOF record. Tape marks are used to separate two sets of data or to separate sets of data from labels.

Symbol	Meaning
IS.SUC	Operation successfully completed.
IE.ABO	Operation aborted.
IE.BBE	Tape format error.
IE.BYT	Odd virtual address or odd byte count.
IE.DAA	Device is already attached.
IE.DAO	Record length error on read. Record exceeded stated buffer size in which case the final portion of the record is not read.
IE.DNA	Device was not attached (Detach failed).
IE.DNR	Device went off-line in the middle of READ/WRITE/WRITE-END-OF-FILE/SKIP functions. This can be the result of a power failure in the middle of an operation or the operator manually turning the unit to off line. This is a serious error as the program cannot be certain that the tape is correctly positioned for subsequent commands.
IE.EOF	An End-of-File character was detected in READ or SKIP function. The tape is left positioned in the gap following the EOF character (or preceding it in the case of backspace).
IE.EOV	Logical end-of-volume (that is, two EOF characters in immediate succession; see Figure 7–4 EOV Handling).
IE.EOT	The unit has sensed the End-of-Tape (EOT) marker while moving in the forward direction. This error will persist until the EOT marker is passed in the reverse direction. (Note that after EOT, 10 feet of tape is provided for writing necessary volume trailer labels.)
IE.FHE	Fatal hardware error. Indicates that the unit may be malfunctioning. IOST+2 contains the actual magtape status register bits at the time of the error.
IE.IFC	Invalid function code specification.
IE.PRI	User task did not have proper access rights.
IE.SPC	Validation error in Read/Write functions.
IE.SRE	Send/request failed.
IE.VER	Parity error in Read/Write or Write end-of-file functions.
IE.WLK	Hardware (or software) write lock error in Write or Write-end-of-file functions.

7.5 UNIBUS Mapping Registers

When running on an 11/44, the MM.... handler uses the mechanism of dynamic UMR allocation that is described in Section 4.4.

When used for controlling UNIBUS magnetic tape subsystems (for example, TU10), the MT.... handler initially requests eight UMRs. If it cannot obtain eight, the handler takes as many as are available. Both read and write operations will be rejected if insufficient UMRs are available to cover the transfer length.

8 Laboratory Peripheral System Handler (LPS11)

8.1 LPS11 Functions

The LPS11 Laboratory Peripheral System is a modular, real-time subsystem that includes the following:

- 12 bit analog-to-digital converter, with sample and hold circuitry and an 8-channel multiplexer
- Programmable real-time clock for measuring and counting intervals or events
- Display controller to display data in a 4096 by 4096 dot matrix
- Digital input/output option (16 digital points and programmable relays)

Built in a compact size and designed for easy interfacing with outside instrumentation, the LPS11 is suited to a variety of applications, including biomedical research, analytical instrumentation, data collection and reduction, monitoring, data logging, industrial testing, engineering, and technical education.

The LPS11 handler allows many users to share access to the basic LPS11 facilities listed below; therefore, the LPS11 device cannot be attached to an individual task. The LPS11 handler allows time-based sampling to be initiated on one channel while sampling is in progress on other channels. Thus, experiments can be started at any time, independent of the current laboratory work load.

8.1.1 Digital I/O

To support the LPS11 Digital I/O module (LPSDR) a bit mask word is initialized in the LPS PUD entry during system generation. Each bit in the mask corresponds to a bit in the digital I/O word, which in turn corresponds to one of sixteen channels.

A mask bit value of 0 specifies that the digital I/O word bit is never to be set to 1 by the LPS11 handler. A mask bit value of 1 specifies that the digital I/O word bit is set only when sampling is being done on the corresponding channel as the result of a user request. That is, a value of 1 in mask bit 4 means that bit 4 of the output word is set only when sampling is in progress on LPS-11 channel 4.

A digital I/O bit set to 1 is a signal to an external device that the LPS11 is ready to accept data on the channel. For this reason, all external devices sending data to the LPS11 must be under the control of the central processor that tells the devices when to start and stop.

When a mask bit is equal to 1, sampling of data on the corresponding channel is restricted to one user program to prevent random changes from being made to the bit. When the bit is 0, multiple users are allowed to read data from the channel since it is impossible for a user program to damage any other simply by reading data.

NOTE: If the Digital I/O option (LPSDR) is not present on an LPS configuration, the mask word in the PUD must be 0.

8.1.2 Real-Time Clock

The Real-Time Clock module (LPSKW) is set to mode 1 (repeated interval mode) with interrupts enabled and a base rate of 10kHz. A preset value of -10 remains constant in the buffer throughout LPS11 handler operation, which means that once every millisecond the handler receives an interrupt from the clock. Therefore, the highest sampling rate on any channel is 1000 points per second.

8.1.3 12-Bit A/D Converter

Sampling of channels through the 12-bit A/D Converter (LPSAD-12) is permitted on any channel whose channel number is less than or equal to the maximum number of channels in the LPS configuration. This value is stored in the PUD entry created by system generation.

When an interrupt occurs for the real-time clock module, the internal LPS clock queue is examined. If any samples are due to be taken, the A/D conversion is initiated on the appropriate channel.

When the clock node has been completely processed, the A/D status word is read. If the error bit is set, a value of -2 is put into the user's buffer for that sample. If the DONE bit is not set, an A/D timeout is indicated, and a value of -1 is put into the user's buffer. If the DONE bit is set, the A/D value is put into the user's buffer and the clock queue is again examined for more samples to be taken.

8.2 System Generation Options

At system generation, the user can specify the following characteristics which result in bit settings in the device PUD:

1 The number of A/D channels in the low-order byte of characteristics word 2.

2 Whether the gain ranging option (LPSAM-SG) is present (bit 15 in characteristics word 2 is set if present).

3 Whether the D/A option (LPSVC or LPSDA) is present (bit 14 of characteristics word 2) and how many D/A channels (low 5 bits of the high-order byte of characteristics word 2).

4 The polarity of each A/D channel (unipolar or bipolar). For each channel, the corresponding bit is set in characteristics word 3 of the PUD if the channel is unipolar; for example, setting bit 0 indicates that channel 0 is unipolar.

Multiple controllers are not supported.

8.3 QIO MACROS

This section summarizes standard and device-specific QIO functions for the LPS11 handler.

8.3.1 Standard QIO Function

The only device independent QIO macro that is valid for the LPS11 is as follows:

```
QIO$ IO.KIL
```

This QIO cancels all queued and in-progress I/O requests.

8.3.2 Device-Specific QIO Functions (Immediate)

All device-specific functions of the QIO macro that are valid for the LPS11 are either immediate or synchronous except for IO.STP (see Section 8.3.4). Each immediate function performs a complete operation, whereas each synchronous function simply initiates an operation. Table 8–1 lists the immediate functions.

Table 8–1 Device-Specific QIO Functions for the LPS11 (Immediate)

Format	Function
QIO$C IO.LED,...,<int,num>	Display number in LED lights
QIO$C IO.REL,...,<rel,pol>	Latch output relay
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.

IO.LED

This function displays a 16-bit signed binary integer in the light-emitting diode (LED) lights. The number is displayed as five nonzero-suppressed decimal digits that represent the magnitude of the number. A minus sign precedes a negative number. LED digits are numbered from right to left, starting at 1.

The number can be displayed with or without a decimal point. If the parameter num is a number from 1 to 5, the corresponding LED digit is displayed with a decimal point to the right of the digit; otherwise, no decimal point is displayed.

IO.REL

This 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 handler imposes no delays when executing this function. Thus, it is the responsibility of the user to ensure that adequate time has elapsed between the opening and closing of a relay.

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. The resulting value is returned in the second word of the I/O status word.

The operation performed is:

```
RETURN VALUE=MASK.AND.INPUT REGISTER
```

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>
```

8.3.3 Device-Specific QIO functions (Synchronous)

Table 8–2 lists the synchronous, device-specific functions of the QIO macro that are valid for the LPS11.

Table 8–2 Device-Specific QIO Functions for the LPS11 (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
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 LPS11 clock ticks between samples or data transfers, as appropriate.
- bufs - is the number of data buffers to transfer.
- chna - is the A/D conversion specification (byte 0 - starting A/D channel number, which must be in the range 0-63. If the gain ranging option is present the channel number must be in the range 0-15 and bits 4 and 5 specify the gain code. Byte 1 - number of consecutive A/D channels to be sampled, which must be in the range 1-64).
- chnd - is the D/A output channel specification (byte 0 - starting D/A channel number, which must be in the range 0-9; byte 1 - number of consecutive channels to output, which must be in the range 1-10).
- mask - is the mask word.

The following subsections describe the functions listed above.

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. The auto gain-ranging algorithm causes a channel to be sampled at the highest gain at which saturation does not occur.

Sampling can 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 handler fills the other half of the buffer.

The subfunction modifier bits are identical to those described in "IO.HIS"; in addition, setting bit 3 to 1 requests auto gain-ranging. 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 auto gain-ranging is used, the LPSAM-SG hardware option must be present and specified at system generation. If the gain-ranging option is present and auto gain-ranging is not specified in bit 3 of the subfunction code, 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 through 11 and the gain code, as shown below in bits 12 through 15:

Code	Gain
0000	1
0001	4
0010	16
0011	64

If the LPSAM-SG option is present, the baud pass filter jumpers must not be clipped. Also, each channel must have been defined as unipolar or bipolar at system generation by defining the corresponding bit (for example, bit 0 for channel 0) in characteristics word 3. Setting a bit indicates that the channel is unipolar.

IO.HIS

This 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 increased and Schmitt trigger one is tested. If it has fired, 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 can be started and stopped as described in "IO.ADS". 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

Bit	Meaning
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)

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 can be started or stopped as described in "IO.ADS". 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 LPSVC option. D/A converters 2 through 9 correspond to the LPSDA external D/A option.

The subfunction modifier bits are identical to those described in "IO.HIS".

IO.MDI

This function provides the capability to read data that is qualified by a mask word from the digital input register at precisely timed intervals. Sampling can be started and stopped as described in "IO.ADS". 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 "IO.HIS".

IO.MDO

This function writes data qualified by a mask word into the digital output register at precisely timed intervals. Output can be started and stopped as described in "IO.ADS". All output is double-buffered with respect to the user task.

The subfunction modifier bits are identical to those described in "IO.HIS".

8.3.4 Device-Specific QIO Function (IO.STP)

Table 8–3 lists the device-specific IO.STP function of the QIO macro, which is valid for the LPS11.

Table 8–3 Device-Specific QIO Function for the LPS11 (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).

IO.STP

IO.STP stops a single synchronous request that is in progress. It is unlike IO.KIL in that it only cancels the specified request. IO.KIL cancels all requests.

8.4 FORTRAN Interface

The FORTRAN-callable subroutines, described in this section provide FORTRAN programs with access to the LPS11. Some of these routines can be called from FORTRAN as either subroutines or functions. All are reentrant and can be placed in a resident library. They are included in SYSLIB in the distributed version of IAS.

8.4.1 The isb Status Array

The isb (I/O status block) parameter is a 2-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 handler 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 its contents varies depending on the FORTRAN call that has been executed. Table 8–4 lists certain general principles that apply. The sections describing individual subroutines provide more details.

Table 8–4 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 illegal time or buffer value
3 < isb(1) < 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.

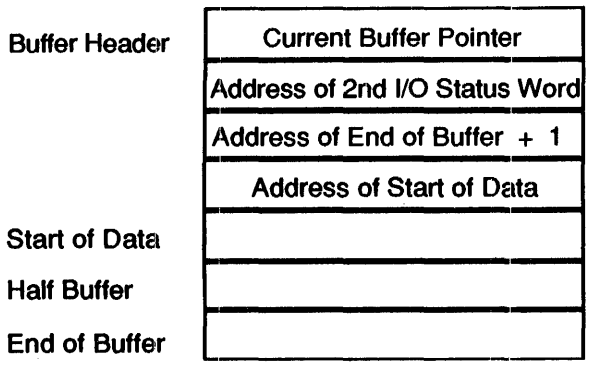
8.4.2 Synchronous Subroutines

RTS, DRS, HIST, SDO, and SDAC are FORTRAN subroutines that initiate synchronous functions. When they are used, the LPS11 handler 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 an even number of words and no smaller than six words. An even length is required so that the buffer is exactly divisible into half buffers.

The LPS11 handler performs double buffering within the half buffers. Each time the handler 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.

Figure 8–1 Synchronous Subroutines



If the user task does not respond quickly enough, a data overrun may result. This occurs if the handler attempts to put another data item in the user buffer when no space is available (that is, the buffer is full of data) or if the handler attempts to obtain the next data item from the user buffer when none is available (that is, the buffer is empty).

All synchronous functions may be initiated immediately or when a specified digital input point is set (that is, a start button is pushed).

They can 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 (that is, a stop button is pushed). A digital output point can optionally be set at the start of a synchronous function. It can be used, for example, as a signal to start a test instrument.

8.4.3 FORTRAN Subroutine Summary

Table 8–5 lists the FORTRAN interface subroutines for the LPS11. S and F indicate whether they can be called as subroutines or functions.

Table 8–5 FORTRAN Interface Subroutines for the LPS11

Subroutine	Function
ADC	Read a single A/D channel (F,S)
ADJLPS	Adjust buffer pointers (S)
ASLSLN	Assign a LUN to LSO: (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)
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)
LPSTP	Stop an in-progress synchronous function (S)
PUTD	Put data into a synchronous function output buffer (S)
RELAY	Latch an output relay (S)
RTS	Initiate synchronous A/D sampling (S)
SDAC	Initiate synchronous D/A output (S)
SDO	Initiate synchronous digital output (S)

The following subsections briefly describe the function format of each FORTRAN subroutine call.

8.4.4 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 can be converted at a specific gain or the handler can select the best gain; that is, 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:

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	Autogain-ranging (handler selects gain that provides most significance)
1	1
2	4
3	16
4	64

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

The isb array has the standard meaning described in Section 8.4.1.

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 8.5.3). Otherwise, this value is a floating-point number calculated from the following formula:

```
var = (64 * converted value) / conversion gain
```

8.4.5 ADJLPS: Adjusting Buffer Pointers

The ADJLPS FORTRAN subroutine adjusts buffer pointers for a buffer that the LPS11 handler 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 can be used. In some cases, however, it is useful to leave data in the buffer until processing is complete. The user program can process the data directly and 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 ensure that the program stays in synchronization with the LPS11 handler. If the program loses its position with respect to the handler, the function must be stopped and restarted. An attempt to overadjust causes 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 8.4.1.

8.4.6 ASLSLN: Assigning a LUN to LSO:

The ASLSLN FORTRAN subroutine assigns a logical unit number (LUN) to the LPS11. It must be called before execution of any other LPS11 FORTRAN function or subroutine. Subsequent calls to other interface routines then implicitly refer to 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 the LSO:
- isb - is a 2-word integer array to which the subroutine status is returned.

The isb array has the standard meaning described in Section 8.4.1.

8.4.7 CVSWG: Converting a Switch Gain A/D Value to Floating-Point

The CVSWG FORTRAN function converts an A/D value from a synchronous A/D sampling function to a floating-point number. Each data item returned by the LPS11 handler consists of a gain code and converted value packed in a single word (see "IO.ADS"). This form is not readily usable by FORTRAN, but is much more efficient than converting each value to floating-point in the LPS11 handler. This routine unpacks the gain code and value, then converts the result to a floating-point number. It can be conveniently used in conjunction with the IRDB routine (see Section 8.4.12).

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

8.4.8 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 can be started or stopped as for RTS (see Section 8.4.17) and all input is double-buffered with respect to the user task. Data can be sequentially retrieved from the data buffer via the IRDB routine (see Section 8.4.11), or the ADJLPS routine (see Section 8.4.5 can 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 or 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	LPS11 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 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.

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.

8.4.9 HIST: Initiating Histogram Sampling

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 can be started and stopped as for RTS (see Section 8.4.17) 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 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 samples. The first word specifies the interval units as follows:

irate(1)	Unit
1	LPS11 clock ticks
2	Milliseconds
3	Seconds

Irate(1)	Unit
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 8.4.1.

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.

8.4.10 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 8.4.1.

When the function form of the call is used, the value of the function is the same as that returned in ival.

8.4.11 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 8.4.1.

When the function form of the call is used, the value of the function is the same as that returned in newval.

8.4.12 IRDB: Reading Data from an Input Buffer

The IRDB FORTRAN subroutine or function retrieves data sequentially from a buffer that the LPS11 handler 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.

8.4.13 LED: Displaying in LED Lights

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 nonzero-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 can 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 8.4.1.

For example, the following call

```
CALL LED (-55,2)
```

would cause -0005.5 to be displayed in the LED lights.

8.4.14 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 apecifies a buffer that was previously specified in a synchronous initiation request.

8.4.15 PUTD: Putting a Data Item into an Output Buffer

The PUTD FORTRAN subroutine puts data sequentially into a buffer that the LPS11 handler 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.

8.4.16 RELAY: Latching an Output Relay

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 (0 for relay one, 1 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 8.4.1.

8.4.17 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 causes the channels to be sampled at the highest gain at which saturation does not occur.

Sampling can be started when the interface subroutine is called or when a specified digital input point is set. A digital output point can optionally be set when sampling is started. Sampling can 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 handler fills the other half of the buffer. Data can be retrieved sequentially from the data buffer via the IRDB routine (see Section 8.4.11), or the ADJLPS routine (see Section 8.4.5) can 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

- 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	LPS11 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).
- nchan - specifies the number of A/D channels to be sampled (must be between 1 and 64).
- 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 isb parameter has the standard meaning described in Section 8.4.1.

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.

8.4.18 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 can be started and stopped as for RTS (see Section 8.4.17 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 8.4.15) can be used to put data values sequentially into the output data buffer. The ADJLPS routine (see Section 8.4.5) can 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 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 D/A outputs. The first word specifies the interval units as follows:

Irate(1)	Unit
1	LPS11 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).
- nchan - specifies the number of D/A channels to be written into (must be between 1 and 10).
- 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 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 8.4.1.

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.

8.4.19 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 8.4.17) 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 8.4.15) can be used to put data values sequentially into the output data buffer. The ADJLPS routine (see Section 8.4.5) can 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	LPS11 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 only if a function selection value of 32 has been added into imode.

The isb parameter has the standard meaning described in Section 8.4.1.

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.

8.5 Status Returns

The error and status conditions listed in Table 8–6 are returned by the LPS11 handler described in this chapter.

Table 8–6 LPS11 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 cancelled (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.
IE.BYT	Byte-aligned buffer specified
	Byte alignment was specified for a data buffer but only word alignment is legal for the LPS11. Alternately, the length of a buffer is not an even number of bytes.
IE.DAO	Data overrun
	For the LPS11, the handler 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 the LPS11, this code is returned if a device timeout 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 (that is, 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.
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.
IE.ONP	Option not present
	An option dependent 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 the LPS11, a checkpointable task attempted to execute a synchronous sampling function.
IE.RSU	Resource in use

Table 8–6 (Cont.) LPS11 Status Returns

Code	Reason
	A resource needed by the function requested in the QIO directive was being used (see Section 8.5.1).
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 8.5.4.

8.5.1 IE.RSU

IE.RSU is returned if a function requests a resource that is currently being used. The requesting task can 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 LPS11 are each considered a single resource:

Resource	When Shareable
The A/D converter and clock	Always shareable.
Each bit in the digital output register	Never shareable.
Each bit in the digital input register	Always shareable 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 shareable with another such function.

Each resource is allocated on a first-come-first-served basis (that is, when a conflict arises, the most recent request is rejected with a status of IE.RSU).

8.5.2 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 8–7 lists the contents of the second word of the status block, on successful completion for each LPS11 function.

Table 8–7 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 cancelled
IO.LED	Zero
IO.REL	Zero
IO.SDI	Masked value read from digital input register
IO.SDO	Updated value written into digital output register
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. LPS11 handler functions 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	No I/O status block, illegal digital
IO.MDA	I/O point number, or illegal channel number.
IO.HIS	No I/O status block or illegal
IO.MDI	digital I/O point number.
IO.MDO	

8.5.3 IO.ADS and ADC Errors

While IO.ADS or the ADC FORTRAN subroutine is converting a sample, two error conditions can 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.

8.5.4 FORTRAN Interface Values

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

Table 8–8 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

8.6 Programming Hints

This section contains information on important programming considerations relevant to users of the LPS11 handler described in this chapter.

8.6.1 The LPS11 Clock and Sampling Rates

The basic LPS11 clock frequency (count rate) for all synchronous functions is always 10 KHZ.

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. 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 2 KHZ is possible, but not recommended.

8.6.2 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).

8.6.2.1 Buffer Management

The buffer unload protocol for synchronous input functions is described below. The user constructs a 5-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 for the 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 handler 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 handler has a buffer write pointer. The user and the handler share the second I/O status word, which is the number of data words in the buffer that contain data.

Each time the handler attempts to put a sample value into the buffer, it increments the contents of 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 handler's write pointer and the writer 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 activated, it executes the following code:

```
5$:   Clear   efn       ;
10$:  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$:  TST     IO$B      ;REQUEST TERMINATED?
      BNE     40$       ;IF NE YES
      Waitfor  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 handler a read pointer. The entire buffer must be full when the request is executed.

8.6.3 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 WAITFOR 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
```

In both the above examples, care is taken to ensure that the program stays in synchronlization with the LPS11 handler, the function must be stopped and restarted since this is the only way to recover. Caution should a be exercised to ensure that the above program sequence is used to avoid a possible loss of data.

9 Card Reader Handler Tasks

9.1 Devices Supported

The card reader handler tasks support the CR11 or the CD11 punched card readers or the CM11 mark sense reader. There is a device handler task for the CD11 hardware (file CRNP.TSK) and another device handler task to handle either the CR or CM hardware (file CRBR.TSK). The selection of hardware to be supported by the CR/CM handler is determined at assembly time by specifying the hardware type in an assembly statement. The CD handler task services the CD11 hardware only. The card reader handler task is a single controller handler supporting any card reader assigned to CRn (where n is in the range 0 - 7). A copy of the card reader handler task is required for each card reader supported.

9.2 Card Reader Functions

The card reader handler provides the following types of service to the user:

1 Read cards in DEC026 format and translate to ASCII,

2 Read cards in DEC029 format and translate to ASCII,

3 Read cards in the binary format described in Section 9.3.2.

The user has the option of specifying which of the functions he desires at assembly time.

Specifically, with a conditional assembly he can specify the types of translation to be allowed, namely:

1 Read only 026 codes,

2 Read only 029 codes,

3 Read both types of code with 026 as default,

4 Read both types of code with 029 as default.

If no assembly time specification is made, the handler is built to do the following:

1 Read both 026 and 029 code with 029 as default.

2 Prohibit binary reads.

9.3 Data Formats

Data is interpreted as being in either alphanumeric or binary format.

9.3.1 Alphanumeric Format

The translation from 026 or 029 card codes to ASCII is performed as specified in Table 9–1.

9.3.2 Binary Format

Binary data is not translated. After data is first converted to a packed form, it is transferred to the user exactly as read; that is, each four columns = three words. Figure 9–1 below shows only the first 4 columns on the card. This pattern is repeated for each set of 4 columns read.

Card Column	Word 1	Word 2	Word 3
1	bits 15-4		
2	bits 3-0	bits 15-8	
3		bits 7-0	bits 15-12
4			bits 11-0

Figure 9–1 Binary Data Format 48 Bits (3 words, 4 card columns)

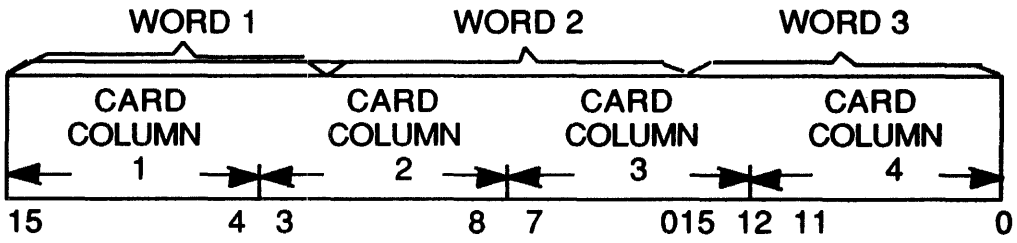


Table 9–1 PDP-11 Punched Card Codes

Character	Non Parity ASCII	DEC029	DEC026
[	173	12 0	12 0
]	175	11 0	11 0
SPACE	040	NONE	NONE
!	041	12 8 7	12 8 7
"	042	8 7	0 8
#	043	8 3	0 8
$	044	11 8 3	11 8 3
%	045	0 8 4	0 8 7
AND	046	12	11 8 7
'	047	8 5	8 6
(	050	12 8 5	0 8 4
)	051	11 8 5	12 8 4
*	052	11 8 4	11 8 4
+	053	12 8 6	12
,	054	0 8 3	0 8 3
-	055	11	11
.	056	12 8 3	12 8 3
/	057	0 1	0 1
0	060	0	0
1	061	1	1
2	062	2	2
3	063	3	3
4	064	4	4
5	065	5	5
6	066	6	6
7	067	7	7
8	070	8	8
9	071	9	9
:	072	8 2	11 8 2
;	073	11 8 6	0 8 2
<	074	12 8 4	12 8 6
=	075	8 6	8 3
>	076	0 8 6	11 8 6
?	077	0 8 7	12 8 2
@	100	8 4	8 4
A	101	12 1	12 1

Table 9–1 (Cont.) PDP-11 Punched Card Codes

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

9.4 Run Time Service

The user can invoke either alphanumeric or binary service via the QIO Directive. The specific type of alphanumeric translation, 026 or 029, may be defined by the code in the control card that the user must place in front of each deck of cards (for example, 029 control card for an 029 deck, 026 control card for an 026 deck), and an end of file card.

If the READ BINARY function code is not issued, the handler assumes alphanumeric mode. In cases where the handler is assembled for both 026 and 029 cards, the type of translation shall be the last mode invoked, unless the handler is directed otherwise by a special punch in card column 1 of the control card.

Requesting a binary read causes an error return, IE.IFC, if this capability was not incorporated at assembly time.

9.5 Control Characters

The card reader handler task is sensitive to certain control characters. Control characters, when recognized, are never transferred to the user's buffer nor are they included in the word count. The control set consists of the following multipunches:

Punches in card column 1	Meaning
12-11-0-1-6-7-8-9	End of file ASCII mode. In binary mode these punches must be in card columns 1 through 8.
12-0-2-4-6-8	029 codes follow.
12-2-4-8	026 codes follow.

9.6 I/O Functions

I/O requests serviced by the card reader handler are issued via the QIO$ system macro with arguments specified in the following format:

```
QIO$ fc,lun,ef,pri,iosb,ast,[<stadd,size>]
```

where fc can have one of the following values:

Symbol	Meaning
IO.ATT	Attach
IO.DET	Detach
IO.RLB	Read Logical Block
IO.RVB	Read Virtual Block
IO.RDB	Read Binary

The two parameter words bracketed by left and right angle (<>) brackets apply only to Read Virtual Block and Read Logical Block. They are optional; however, if the parameter words are specified they must be delimited by the angle brackets. They have the following meanings:

- stadd - starting address of the buffer
- size - size of the buffer

9.7 Recovery Procedures

Recovery procedures fall into two categories:

1 Device errors,

2 Power fail.

9.7.1 Device Errors

No attempt is made to flag errors on incoming data, that is, on illegal punching. There are two forms of hardware error message, which are output to the operator's console.

Hardware error message	Action
***CRn—NOT READY	Place cards in the input hopper and press the reset switch.
***CRn—READ ERROR, CHECK HARDWARE STATUS	If the only error indicated is PICK CHECK, press the RESET switch. If the only error indicated is READ check and the CRBR handler is being used, retrieve the last two cards read and place them in front of the cards remaining in the input hopper. Press the RESET switch. Otherwise, place the last card read immediately in front of remaining cards in input hopper. Press RESET switch. When the CRNP handler (see Section 9.1) is being used, pressing the STOP switch while a card read order is in progress causes this error message. Refeed last card read as just described, and press RESET switch. The error message is repeated. Now press the STOP switch and then the RESET switch.

The request remains pending unless the system indicates that the operator cannot be notified. In this case, the request is terminated with a status of IE.DNR.

9.7.2 Power Failure Recovery

If power fails, the program checks to see if a card read is in progress. If so, a hardware (device) error is simulated.

For either device error recovery or power failure recovery the operator must reinsert the last card read and press reset to restart the reader.

9.8 CR Status Returns

IOST contains a code indicating the disposition of the QIO request. These codes are symbolized as shown below.

Symbol	Meaning
IS.SUC	Successful completion
IE.BAD	Invalid parameters
IE.IFC	Invalid function code (returned only when handler was not assembled to process binary data and a binary read was requested)
IE.DNR	Device not ready

Symbol	Meaning
IE.SPC	Part of buffer out of user space
IE.EOF	End of file or read
IE.PRI	User does not have directive privilege
IE.DNA	Device not attached
IE.DAA	Device already attached

See Appendix A.

9.9 UNIBUS Mapping Register (UMR) Allocation

On PDP-11/70 processor systems with more than 124K of memory, the handler requests allocation of one UMR for DMA access. The UMR is allocated when the handler is loaded into memory and deallocated when the handler exits. If none is available, the handler declares itself nonresident and exits.

10 Line Printer Handler

10.1 Printer Functions

The line printer handler task is a single controller handler, supporting any line printer assigned to LPn, where n is in the range 0-7. The line printer models supported are listed in Table 10–1.

Table 10–1 Line Printer Models

Model	Column Width	Characters
LA180	132	96 (medium speed)
LP11-F	80	64
LP11-H	80	96
LP11-J	132	64
LP11-K	132	96
LP11-R	132	64 (heavy duty, high speed)
LP11-S	132	96 (heavy duty, high speed)
LP11-V	132	64
LP11-W	132	96
LS11	130	62 (medium speed)
LV11	132	96 (electrostatic printer-plotter)

The line printer handler task does not support the special features of the LV11 plotter mode.

The line printer handler task LP.... is written as a multi-user task. This means that a separate copy of the task runs for each printer in the system. The read-only part of the handler code is shared between all copies.

10.2 System Generation Options

Some types of line printer (for example, LP11-V) need their buffer be cleared before they can be turned off-line. In these cases, a carriage return code must be explicitly output when one is implied; there is no wasted print cycle. These types of printer must be specified as LS-types at system generation (See the description of the DEV directive in the *IAS System Generation and Startup Guide* for more detail).

Other types (for example, LP11-R) clear their own buffer when they are turned off-line. In these cases, a carriage return code need be output only when overprinting is required. Further, redundant carriage return codes cause wasted print cycles on these types of printer, and they should be specified as LP-types at system generation.

For line printers specified as upper case only at system generation, the handler converts any lower case characters to their upper case equivalents.

If an upper case only printer is specified as upper and lower case at system generation, the handler will not perform this case conversion. Typically, lower case characters will appear as blanks on the printer.

10.3 Function Codes

I/O requests serviced by the line printer handler are issued via the QIO$ system macro with arguments specified in the following format:

```
QIO$[S] fc,lun,ef,pri,iosb,ast[,<stadd,size,vfc>]
```

Function codes are:

- IO.WVB - Write Virtual Block (print buffer contents)
- IO.WLB - Write Logical Block (print buffer contents)
- IO.ATT - Attach Printer for Reserved usage
- IO.DET - Detach Printer

The three parameter words bracketed by left and right angle brackets (<>) apply only to Write Virtual Block and Write Logical Block. When used with these functions, the parameter words must be delimited by the angle brackets. They have the following meanings:

- stadd - starting address of the buffer
- size - size of the buffer
- vfc - vertical format control character. This has one of the values shown in Table 10–2. Note that the IAS spooler will always translate line feeds into null buffer writes to ensure that the line feed appears correctly whatever the printer type.

Table 10–2 Vertical Format Control Characters

Octal Value	Character	Meaning
040	blank	SINGLE-SPACE - The handler outputs a line feed, outputs the contents of the buffer, and (for an LS-type printer) outputs a carriage return. Normally, printing immediately follows the previously printed line.
060	0 (zero)	DOUBLE SPACE - The handler outputs two line feeds, outputs the contents of the buffer, and outputs a carriage return. Normally, a blank line is output followed by buffer contents printed two lines below the previously printed line.
061	1 (one)	PAGE EJECT - The handler outputs a form feed, outputs the contents of the buffer, and outputs a carriage return. Normally, the contents of the buffer are printed on the first line of the next page.
053	+ (plus)	OVERPRINT - The handler outputs a carriage return, and outputs the contents of the buffer. Normally, the contents of the buffer are printed on the same line as the previously printed line.
044	$ (dollar sign)	PROMPTING OUTPUT - The handler outputs a line feed (except for an LS-type printer), and outputs the contents of the buffer. This mode of output is intended for use with a teleprinter, where a prompting message is output, and input is read (echoed) on the same line. Note that the line printer hardware treats a line feed as a carriage return, line feed (except for LS11).

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

Table 10–2 (Cont.) Vertical Format Control Characters

Octal Value	Character	Meaning
000	null	INTERNAL VERTICAL FORMAT - The handler outputs the contents of the buffer, and does not output a vertical control character. In this mode, more than one line of guaranteed contiguous output may be printed per I/O request queued.

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

10.4 LP Status Returns

IOST contains a code indicating the disposition of the QIO request. These status return codes for the line printer handler task are symbolized as shown below.

Symbol	Meaning
IS.SUC	Successful completion
IE.IFC	Invalid function code
IE.SPC	Part of buffer is out of address space
IE.DNA	Device not attached
IE.DAA	Device already attached
IE.DAO	Output truncated (line too long)

See Appendix A for a complete list of I/O status returns.

10.5 Characteristics Words for Line Printer

Section 1.8.1 describes the general use of the four characteristics words in the PUD. For lineprinters the following bits are significant in characteristics word 2:

bit 0	H1.LC	if set printer has lower case
bit 1	H1.LS	if set printer is an LS11

These bits are set during system generation and do not vary.

11 Message Output Handler

11.1 Message Output Handler (MO)

IAS provides a facility for all outputting user-defined messages. Its goals are:

1 To provide a system-wide standard for message output with emphasis on error reporting.

2 To keep the code per task as small as possible and still provide coherent information.

After issuing a message (error or otherwise), the task that requests the message handler can:

1 Continue

2 Be immediately suspended until the operator continues or aborts the task

3 Do further processing, then be suspended as in b

4 Be immediately aborted

The MO handler (task name = MO....), has two message destinations:

1 SYLOG - the system logging device

2 USBUF - a user specified buffer

You can specify one or both.

11.1.1 User Task Interface To MO Handler

Message output is initiated via a QIO Directive to the device MO. The MO task is a device handler, and as such it is a privileged task, with an entry in the PUD called MO. The user commonly issues the QIO directive implicitly via one of the macros MOUT$, MOUT$C or MOUT$S described in Section 11.4.

As with all handlers, a requesting task must assign a LUN to device MO in order to use it. The LUN can be assigned in three ways:

1 The user task can execute the ASSIGN directive to device MO.

2 The user task can reserve a global memory location using the instruction:

```
.MOLUN:: .BLKW 1
```

In this instance, the task builder will put an extra slot in the task's logical unit table, store its number in .MOLUN, and assign the LUN to MO.

3 The user can assign the LUN via the task builder command, as:

```
ASG=MO:n
```

where n is specified as the LUN value.

The macros which can replace explicit use of a QIO to MO are supplied in the system macro file RSXMAC.SML.

11.1.2 String Descriptors

The first parameter of the macros that issue a message using MO is the address of a string descriptor (see Section 11.4). When the message is in the user program, the string descriptor is a pointer to a format string; however, if the message is in a disk file, the string descriptor is a pointer to a filename string.

The string descriptor is two words long. The first word is the length of the format string or the filename string. The second word is the address of the format string or the filename string. Example 11–1 and Example 11–2 show the string descriptor for message format strings included in the user program. Example 11–3 shows the string descriptor pointing to a filename string.

11.1.3 Parameter List

The message handler has the capability of inserting user arguments into a predefined message. A pointer to the parameter list is the second parameter in the macro calls and is the address of a table of sequential arguments to be inserted in the message. The format of the table depends on the code used for constructing the message. See Section 11.3.

11.2 MO Task Operation

After the MO task has dequeued the request node set up by the user QIO Directive, message processing proceeds as follows.

1 The format string is moved from the user task or specified file to the MO task area.

2 The message string is created (see Section 11.3).

3 If a user buffer is specified, a copy of the message is moved there. (See Section 11.5 for format.)

4 The request node is released declaring I/O done. Event flag 30 (decimal) is set only if the user task specified CONTINUE as the following action. Event flag 30 (decimal) is not set if the user task specified SUSPEND in the macro.

If ABORT was specified, the requesting task is terminated by the ABORT directive.

5 If SYLOG was specified as the destination, the message is sent to the system logging device.

Note that the output of a message to the system logging device on behalf of a user task is always asynchronous to that task's execution. If the using task specifies an I/O status block, a return code is placed in it. A positive value in the status block indicates a successful return. A negative value indicates an error return. A successful return to the user means that all processing up to but not including the output to the system logging device was correct. Status returns are defined in Section 11.8.

11.3 Message Construction

Messages are constructed from formatted strings that can be stored either in the user task space or in a disk file, with a fixed length of 64-byte records. The format string consists of fixed and variable characters. In order to construct a message, the user must supply values to be substituted for the variables. An example of a format string is:

```
ALPHA:%8A
```

The format string is scanned and each character is copied into the message buffer until a % character is encountered. This triggers an interpretation of the next few characters according to the following syntax:

```
% count code
  or
 % V code
  or
  % %
```

where:

- count - is a numeric ASCII string that is converted into a positive decimal integer indicating how many times the action indicated by the code character is to be performed. If not specified, 1 is assumed.
- V - is used to indicate that the count is variable and the next word in the parameter list is interpreted as the count.
- code - is a single letter indicating the action to be performed. The code values are summarized in Table 11–1.
- % - is used to output % to the message text.

In the example, %8A is interpreted to mean:

Move 8 characters from the buffer whose address is the next item in the parameter list.

In the example %VA, the interpretation is:

Get the count, a variable number, from the current position of the parameter list pointer. Increment the pointer and move the variable number of bytes or characters as stated above.

In the example below, the first word of the string descriptor STR1, contains the format string length and the second word contains the format string address. PAR1 is the address of the parameter list.

To pass format string information to MO the user task proceeds as follows:

1 Assume the formatted message was assembled as:

```
STR1:   .WORD   LN1E-LN1   ;LENGTH OF FORMAT STRING
     .WORD   LN1      ;ADDRESS OF FORMAT STRING
LN1:    .ASCIZ  /ALPHA:%8A/
LN1E:
```

2 During program execution a parameter list contains the pointer to the ASCII string.

```
PAR1:   .WORD   ASTR
        .
        .
        .
ASTR:   .ASCII  /ABCDEFGH/
```

3 The user task issues a macro call:

```
MOUT$S  #STR1,#PAR1
```

4 MO responds to the macro call and outputs the following message to the system logging device:

```
****taskname - CONTINUED
ALPHA:ABCDEFGH
```

In Table 11–1, the %nT command (FORTRAN Trace-back Chain) takes two arguments: the trace-back chain listhead and the current statement number. The trace-back chain listhead is the contents of location $NAMC; the current statement number is the contents of location $SEQC.

The steps corresponding to 2 and 3 above are (with n=20):

2 Set up parameter list:

```
PARAM :     .BLKW       2
STRING:     .ASCIZ      /%20T/
```

3 Code and macro call:

```
MOV     $NAMC, PARAM
MOV     $SEQC, PARAM+2
MOUT$S  #STRING, #PARAM
```

Example 11–1 and Example 11–2 show more complex format strings, the macro calls to format them, and the resulting output string. Section 11.5 describes default values that are used to construct the DPB when parameters are omitted.

Table 11–1 Format String Codes

Code	Meaning
%nA	Move n bytes from the buffer whose pointer is taken from the current location of the parameter list.
%nB	Print n bytes as 3-digit octal numbers. The current item in the parameter list gives the address of the address of the first byte. The bytes are separated by spaces.
%nD	Convert n words beginning at the current location in the parameter list. Each word produces a signed zero-suppressed ASCII string (maximum, five digits) that is a decimal representation of the word.
	Each converted digit string is delimited by a tab. If the sign is omitted, + is implied.
%nO	Convert n words beginning at the current location in the parameter list. Each word produces a signed, zero-suppressed ASCII string (maximum, five digits) that is an octal representation of the word.
	Each converted digit string is separated by a tab. Omitting the sign implies +.
%nP	Same as %nO except that each word converts to an unsigned 6-digit octal string.
%nR	Convert n words beginning at current location in the parameter list. Each word is interpreted as a RADIX-50 word and converts to three ASCII characters. No spaces are inserted between converted 3-character strings.
%nF	Insert n form feed characters into the output string. Insertion of form feeds does not imply start of a new record.
%nL	Start a new record in the output buffer. This code implies carriage return, line feed (a CR,LF) when output is to the system logging device. If n is greater than 1, additional 0 byte records are inserted in the output buffer.
%nN	Insert n line terminators (CR,LF) in the output string. The current record is not terminated.

Table 11–1 (Cont.) Format String Codes

Code	Meaning
%nT	Trace the FORTRAN traceback chain through n links or until the link pointer is 0, whichever comes first.
%nX	Insert n filename strings in the output buffer beginning at the current location in the parameter list. A filename is described by five parameter words: • Words 0-2 - Radix-50 filename; • Word 3 - Radix-50 file type; • Word 4 - Binary version number. The filename is printed in standard syntax and trailing blanks are suppressed. If the version number is 0, it is not printed. Each converted filename is delimited by a space.

11.3.1 Message File

Error messages can be stored in a file. In this case the string descriptor in the macro call refers to a filename string instead of a format string. The default filename type is .MSG. In the example below, R1 contains the address of the parameter list. R2 contains the record number of the format string within the file.

Example 11–1 Example Using Counts in the Format String

```
        MOUT$S #STR1,#PAR1
          .
          .
          .
;
; STRING DESCRIPTOR
;
STR1:   .WORD LN1E-LN1     ;LENGTH OF FORMAT STRING
        .WORD LN1          ;ADDRESS OF FORMAT STRING

;
; FORMAT STRING
;
LN1:    .ASCIZ /ALPHA:%10A,DEC:%3D,OCT:%4O/
LN1E:

;
; PARAMETER LIST
;
PAR1:   .WORD ASTR         ;POINTER TO ASCII STRING

        .WORD 123.         ;ARGUMENTS TO BE USED IN
        .WORD 456.         ;DECIMAL CONVERSION
        .WORD 789.

        .WORD 111          ;ARGUMENTS TO BE USED IN
        .WORD 222          ;OCTAL CONVERSION
        .WORD 333
        .WORD 444

ASTR:   .ASCII /ABCDEFGHIJ/

.MOLUN::.BLKW  1           ;LUN WILL BE ASSIGNED BY
                  ;THE TASK BUILDER
```

The resulting message is as follows:

```
****task name - CONTINUED
ALPHA:ABCDEFGHIJ,DEC:123 456 789,OCT:111 222 333 444
```

Example 11–2 Example Using V in the Format String

```
        MOUT$S #STR2,#PAR2
          .
          .
          .
;
; STRING DESCRIPTOR
;
STR2:   .WORD  LN2E-LN2     ;LENGTH OF FORMAT STRING
        .WORD  LN2          ;ADDRESS OF FORMAT STRING

;
; FORMAT STRING
;
LN2:    .ASCIZ /ALPHA:%VA,DEC:%VD,OCT:%VO/
LN2E:

;
; PARAMETER LIST
;
PAR2:   .WORD  10.          ;
        .WORD  ASTR         ;POINTER TO ASCII STRING

        .WORD  3            ;
        .WORD  123.         ;ARGUMENTS TO BE USED IN
        .WORD  456.         ;DECIMAL CONVERSION
        .WORD  789.

        .WORD  4            ;
        .WORD  111          ;ARGUMENTS TO BE USED IN
        .WORD  222          ;OCTAL CONVERSION
        .WORD  333
        .WORD  444

ASTR:   .ASCII /ABCDEFGHIJ/

.MOLUN::.BLKW  1            ;LUN WILL BE ASSIGNED BY
                            ;THE TASK BUILDER
```

The resulting message is as follows:

```
****task name - CONTINUED
ALPHA:ABCDEFGHIJ,DEC:123 456 789,OCT:111 222 333 444
```

Example 11–3 Example of Format From a Disk File

```
      MOUT$S #FILDES,R1,R2
        .
        .
        .
;
;STRING DESCRIPTOR
;
FILDES: .WORD  ENDFIL-FILNAM  ;LENGTH OF FILENAME STRING
    .WORD  FILNAM     ;ADDRESS OF FILENAME STRING

;
; FILENAME STRING
;
FILNAM: .ASCIZ /DK1:[100,100]SAMPLE.MSG;1/
ENDFIL:
```

11.4 Message Macro Descriptions

This section contains an explanation for each of the macros that are supplied for the users of MO. The macros are listed below.

MOUT$
MOUT$C
MOUT$S
MODF$
MOWA$S

The argument descriptions for MOUT$ (Section 11.4.1) apply also to MOUT$C (Section 11.4.2) and MOUT$S (Section 11.4.3). mout$c and MOUT$S have three additional arguments.

Where a suspension is required, whether immediate or deferred, the parameter "act" must be set to SUSPD. See Section 11.4.5 for details concerning suspension. The operator can reply to the suspension by typing the MCR commands

```
CONTINUE tsknam or ABORT tsknam
```

or the DCL commands

```
CONTINUE/MESSAGE tsknam or ABORT/REALTIME tsknam
```

11.4.1 MOUT$

This macro generates the proper Queue I/O DPB for accessing the MO task.

Macro call:

```
MOUT$ str,prm,num,act,dst,buf,siz,iost,lun
```

Argument	Meaning
str	Address of a string descriptor in the user area that in turn points to a format string in the user area or to the dataset specification of the user's file where the record (format string) is found. See "num".
prm	Prm is a pointer to the parameter list. The parameter list is a sequential list of arguments that are used in formatting the message format string.
num	The value of num determines how str is interpreted. If num is less than or equal to 0, the value of str is a pointer to a format string in the task's address space. If num is positive, str is a pointer to an ASCII filename, and num's value is a record number index to the format string in the named file. If num is not specified zero is used by default.
act	The value specified for act dictates what action is to be taken after the error message is formatted. You can specify one of the following string variables:

Argument	Meaning
	• CONT—Continue the task that is requesting the error message, immediately or after further processing. • SUSPD—Suspend, immediately or after further processing, the requesting task until the operator responds (see Section 11.4). The requesting task must also generate a wait on the special event flag (30 decimal) in order to cause the suspension (see Section 11.4.5). • ABORT—Abort the requesting task. If the action is not specified, CONT is used by default. An action value not matching one of those above is assumed to be described in a user-specified action bit pattern. (See Section 11.4.4.)
dst	One of the following string variables is specified to designate the destination of the error message. • SYLOG—System logging device • USBUF—User buffer • SYABUF—System logging device and user buffer If the destination is not specified, SYLOG is used by default. Output to the system logging device is preceded by a taskname/action header line if the user specifies the string SYLOG as the destination argument in the macro call. To suppress the header the user modifies the destination argument by specifying SY$STM in the bit pattern. See Section 11.4.4. USBUF or SYABUF designates that the message will be sent to a user buffer, indentified in the buf parameter (see below). The header line is never returned to the user buffer. A destination value not matching one of those above is assumed to be described in a user-specified destination bit pattern. Note that if act was CONT, output to SYLOG is performed simultaneously with task operation.
buf	Buf points to the user buffer, where the message is sent when destination requires transfer back to the user task.
siz	The size, in bytes, of the user buffer pointed to by buf. (Maximum = 256/minimum = 6 (decimal).)
iost	This parameter contains a pointer to the user task's I/O status block, which is set on QIO completion, to indicate success or failure.
lun	This parameter defines the logical unit number to be used for the message. If the task has defined the global symbol .MOLUN, the task builder initializes the LUN to incorporate the message output LUN by placing the value of the LUN in location .MOLUN. If the user then invokes the MOUT$S macro, .MOLUN is the default LUN parameter. Otherwise, lun must be specified in the macro call.

The following symbols are generated for accessing the DPB at run time. They are logically defined values equal to byte offsets from the start of the DPB to the respective elements:

- M.OLUN—(Length 2 bytes) Logical Unit
- M.OIST—(2) Address of I/O status block
- M.ODST—(1) Destination
- M.OACT—(1) Action
- M.ONUM—(2) Record number
- M.OSTR—(2) Format string descriptor pointer
- M.OPRM—(2) Parameter list pointer
- M.OBUF—(2) User buffer pointer
- M.OSIZ—(2) User buffer size

11.4.2 MOUT$C

The MOUT$C macro generates a Queue I/O DPB in a separate program section named $DPB$$. A monitor call and, if specified, a wait for event flag 30 are generated in the original program section.

Macro call:

```
MOUT$C str,prm,num,act,dst,buf,siz,iost,lun,cs,err,now
```

In addition to the macro arguments described in Section 11.4.1, MOUT$C uses the three arguments described below.

Argument	Meaning
cs	This parameter only appears with MOUT$C. It should specify the name of the current program section. (Refer to the IAS System Directives Reference Manual for an explanation of the $C form of system directives.)
err	If a value is given for err, it specifies a location to be called if the Queue I/O directive fails.
now	If this parameter is null, the WAIT FOR event flag 30 is generated. If NWAIT is specified, the WAIT FOR is not generated.

1.4.3 MOUT$S

The MOUT$S macro generates the code to construct a DPB on the user stack and issue the Queue I/O directive to the message output task (MO). Provision is included for specifying an error address, in case the Queue I/O directive is unsuccessful, and a WAIT FOR on the special event flag.

Macro call:

```
MOUT$S str,prm,num,act,dst,buf,siz,iost,lun,err,now
```

All arguments, with the following exceptions, are expected to be proper symbols for use in MOV and MOVB source fields: act and dst are string variables for which a good comparison with the predefined symbols listed in Section 11.4.1 will cause the proper code generation. The argument for the WAIT FOR (now) expects the string NWAIT if the user does not want a special WAIT FOR

generated after the QIO. The argument err is a user defined address to be called if the QIO fails. See Section 11.4.1 and Section 11.4.2 for argument definitions. %RUNOFF-W-IIF, ^[ignored

11.4.4 User Definition of Action and Destination

It might be desirable to use a DPB that has been defined by one of the MOUT$ macros, and then to change action codes and destinations for different messages at run time. The macro MODF$ is provided to define locally the symbols used to achieve this change.

For example, in the following subroutine assume that on entry, R0 contains the DPB address. R1 contains a flag which, if negative, means that the task is aborted and that the header line (i.e., taskname and action) is printed. If R1 is positive, the task continues and the header line is not printed.

```
        .MCALL   MODF$
        MODF$
;
ERRA:    TST     R1              ;ENTRY POINT
        BMI     FATAL           ;IS THIS A BAD ERROR
        MOVB    #C$ONT,M.OACT(R0)       ;NO SO JUST
                                ;CONTINUE WITH WARNING
        MOVB    #SY$STM,M.ODST(R0)      ;ON SYSTEM
        BR      ERRTN           ;LOGGING DEVICE
;
FATAL:   MOVB    #A$BRT,M.OACT(R0)       ;BAD ERROR SO ABORT
                                ;THE TASK AND ALSO
        MOVB    #SY$STM!HE$ADR,M.ODST(R0)  ;PRINT THE
                                ;HEADER LINE ON
                                ;SYSTEM LOG
ERRTN:   RTS     PC
```

The MODF$ macro does not have parameters in the call. All the macro does is define the following symbols local'

act	
C$ONT	Continue the requesting task
S$USP	Suspend the requesting task
A$BRT	Abort the requesting task

dst	
SY$STM	System logging device
BU$FFR	User buffer
HE$ADR	Include taskname/action header line with output.

Note that, as shown in the example, the destination codes can be combined by the logical OR operator (!).

As was stated in Section 11.4.1, it is possible to suppress the taskname header line. The example above shows how to suppress the header at run time by moving the destination code SY$STM into the DPB using the offset M.ODST. The header can also be suppressed by using MODF$ symbols at assembly time, for example,

```
        .MCALL    MODF$,MOUT$
;
        MODF$
MESDST  = SY$STM!BU$FFR
;
        MOUT$     str,prm,num,act,MESDST,buf,siz,iost,lun
```

The macro MOUT$ sets the destination code of the DPB to the value of the user defined symbol MESDST (since it is not one of the codes listed in Section 11.4.1 and checked for by MOUT$). However, in the example we have defined MESDST to be the inclusive OR of SY$STM and BU$FFR, thus omitting the code for taskname header printing. Hence the message goes to the system logging device and the user buffer without the header line.

11.4.5 Uses of the MO WAIT FOR Macro

The macro MOWA$S (no parameters) is used with MOUT$ to generate an immediate wait till the operator acts in response to the message. It is also used with MOUT$, MOUT$C and MOUT$S when the wait is deferred.

For immediate suspension, use

```
MOUT$
MOWA$S
  or
MOUT$C  with "now" set to null
  or
MOUT$S  with "now" set to null
```

For deferred suspension, use

```
MOUT$
processing
MOWA$S
  or
MOUT$C  with "now" set to NWAIT
processing
MOWA$S
  or
MOUT$S  with "now" set to NWAIT
processing
MOWA$S
```

11.5 Message DPB Format

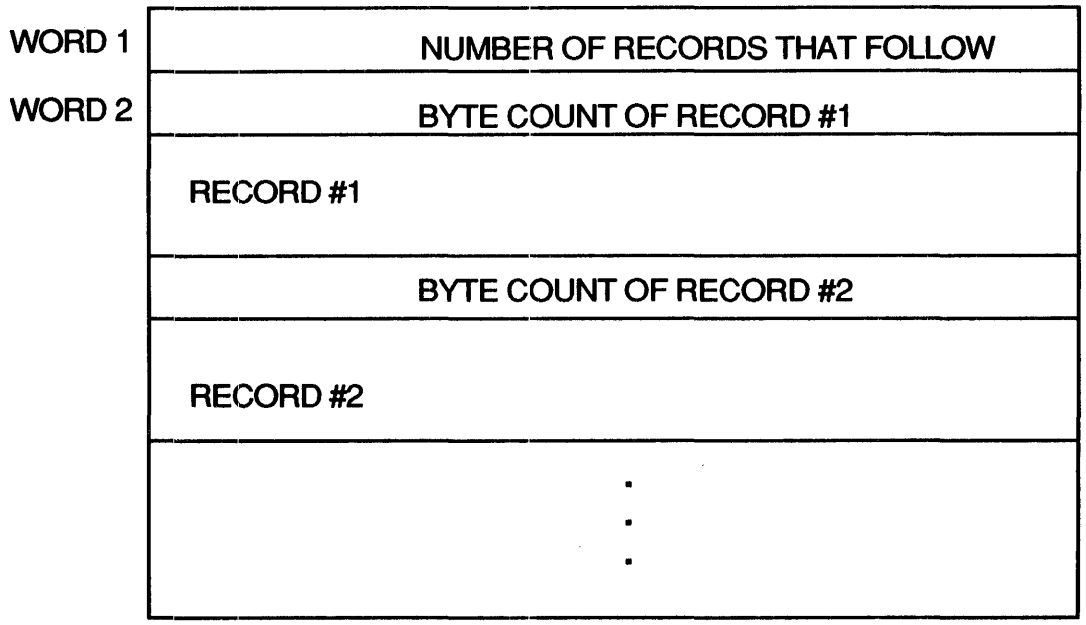
The DPB format for Message Output is shown below with its relation to the MOUT$ macro call. The macro call argument definition has been arranged so that the arguments used most appear first. Note that macro invocation fixes certain values in the DPB.

```
WORD 1:  DIRECTIVE AND DPB SIZE  ;THE SIZE OF THE DPB WILL VARY
                 ; FROM 9-12 WORDS DEPENDING ON
                 ; SPECIFICATION OF PRM, BUF,
                 ; AND SIZ
                 ;
WORD 2:  I/O FUNCTION       ;FIXED TO "WRITE" FUNCTION CODE
WORD 3:  LUN           ;DEFAULT TO 0 IF EXPANDED BY
                 ; MOUT$ or MOUT$C, DEFAULT TO
                 ; ".MOLUN" IF EXPANDED BY MOUT$S.
                 ;
WORD 4:  EFN,PRIORITY       ;FIXED TO EVENT FLAG 30 (DECIMAL),
                 ; PRIORITY 0.
                 ;
WORD 5:  IOST            ;DEFAULT TO 0 (NO STATUS BLOCK)
WORD 6:  AST           ; FIXED TO "NO AST ENTRY"
WORD 7:  ACT,DST         ; ACTION DEFAULTS TO CONT[INUE]
                 ; DESTINATION DEFAULTS TO SYLOG
WORD 8:  NUM           ; MESSAGE NUMBER DEFAULTS TO 0
                 ;
WORD 9:  STR           ;FORMAT STRING POINTER IF NUM
                 ; IS NEGATIVE OR ZERO. FILENAME
                 ; POINTER TO SYSTEM MESSAGE
                 ; FILE IF NUM IS POSITIVE
                 ;
WORD 10: PRM           ;PARAMETER LIST POINTER DEFAULTS
                 ; TO 0 (NO PARAMETER LIST) - FOR-
                 ; MAT STRING IS THE MESSAGE)
WORD 11: BUF           ;USER BUFFER POINTER - DEFAULTS
                 ; TO SMALLER DPB SIZE
WORD 12: SIZ           ;USER BUFFER SIZE - DEFAULTS TO
                 ; 0 IF BUF IS SPECIFIED.
```

If the macro is invoked with the symbol $$$GLB defined, the DPB is not generated, and the symbolic offsets are defined globally.

11.6 Message Format Returned to User Buffer

The message buffer returned to the user is a single transfer of a maximum 512 bytes. The format of the data is:



The L code in the format string is a record terminator and causes the byte count of the current record to be stored as the first word of that record. When necessary, the record is padded with a zero to ensure an even byte count. Each L code in the action string causes an implied CR,LF on the system logging device.

If the repeat count for the code L is greater than 1, then additional records of no bytes are created. The CR,LF is implicit and is not stored in the user buffer. For example, %3L causes the current record to be terminated plus two zero words indicating two records of length zero.

11.7 Error Conditions

The MO task is designed to output messages in spite of error conditions that might arise. Error conditions are accommodated as follows:

1 If destination is not specified, SYLOG is assumed.

2 If a user buffer is specified in destination, but the DPB size indicates that none was specified, or if the size of the buffer was less than six bytes, output is forced to SYLOG.

3 The maximum output buffer size is 256 (decimal) bytes, or the user buffer size (if specified), whichever is smaller. The maximum input buffer size is 64 (decimal) bytes, or the format string size, whichever is smaller.

4 Errors detected during the reading of a format string from a file (CSI, OPEN, GET, CLOSE) cause an error code to be set in the user status block. The contents of the input buffer are printed as is.

5 Errors detected during the processing of a format string cause three question marks to be appended to the message at the point of the error. An error code is set in the user status block. Possible errors are:

   a. Illegal access to user parameter list,

   b. Message buffer overflow,

c. Illegal format directive.

6 The following message is printed on the operator's console if the MO task was unsuccessful in declaring itself a handler task.

```
****MO.... - EXIT****
"DECLARE AND SET" ERROR
```

11.8 MO STATUS RETURNS

The following I/O status returns are made by MO:

- IS.SUC—Successful MO request
- IE.BAD—Invalid format directive encountered, or buffer size less than established minimum, or invalid destination code.
- IE.IFC—Invalid I/O function code.
- IE.SPC—Argument out of user address space.
- IE.DAO—Data overrun (buffer too small for formatted message).
- IE.PRI—I/O privilege error.
- IE.BNM—Bad filename specified.

In addition, error codes can be generated by the File Control Services Routines during an attempt to access a message file.

12 Paper Tape Reader/Punch Handler

12.1 Devices Supported

The high speed paper tape reader and punch handler tasks support the following devices:

Device	Read Speed	Punch Speed
PC11 paper tape reader/punch	300 char/sec	50 char/sec
PR11 paper tape reader	300 char/sec	
(image mode only: no interpretation of data code)		

12.2 Function Codes

The I/O requests serviced by the paper tape reader/punch handler task are issued via the QIO$ system macro with arguments specified in the following format:

```
QIO$  fc,lun,ef,pri,iosb,ast
```

fc can have one of the following values.

Symbol	Meaning
IO.KIL	Cancel Request
IO.ATT	Attach
IO.DET	Detach
IO.RLB	Read Logical Block (paper tape reader only)
IO.WLB	Write Logical Block (paper tape punch only) Read/Write Logical Block passes the tape image to and from the devices, thus the user task will be able to convert between IAS/RSX format and foreign formats as described.

12.3 Tape Leader/Trailer

The paper tape reader handler task attempts to read leaders (nulls) from the paper tape when the device is attached via the QIO function IO.ATT. The paper tape punch handler attempts to punch 100 leaders (nulls) when the device is attached via the QIO function IO.ATT, or punch 100 trailers (nulls) when the device is detached by the QIO function IO.DET. The handler task returns an end-of-tape condition (IE.EOF) with the byte count if a punch failure occurs during a write logical function or if a device not ready condition occurs during a read logical function. Such a condition can be caused by any of the following:

- No tape,
- Reader off line,
- Power low,
- Malfunction after read begins.

12.3.1 Sequential File Device

The high speed paper tape reader and punch handlers distributed with the IAS system process data as sequential block I/O when using FCS or PIP.

You can reconfigure the handlers to use sequential record I/O with ASCII carriage control when performing read and write virtual functions. The paper tape reader/punch is then treated as a sequential file device by the file control services (FCS). All Files-11 files carry the record structure: start of record in word 1, byte count plus 4 in word 2, the actual data records, and the checksum. The checksum is not included in the byte count; however, it is used to verify the calculated checksum of the stripped byte count when the data is read back from the reader handler.

To reconfigure the handlers, edit the source code to include a definition of the symbol ASCFMT for ASCII format I/O. To enable sequential block I/O, define the symbol FILSYS. Both symbols can be defined in the same handler.

The handler source code is supplied on the binaries distribution under UFD [311,14]. The source code also contains instructions for assembling and building the handlers.

If you require ASCII processing for read and write virtual functions you must also ensure that the device characteristics in the handlers' PUD entries are set correctly. Bits 0 and 1 (carriage control and record-oriented device) must be set in characteristics word one. Set these during System Generation or use the MCR OPE command.

12.4 PT Status Returns

IOST contains a code indicating the disposition of the QIO request. These status return codes for the paper tape reader/punch handler tasks are symbolized as shown below. (See Appendix A.)

Symbol	Meaning
IE.IFC	Invalid function code
IE.SPC	Part of buffer is out of address space
IE.DAA	Device already attached
IE.DNA	Device not attached
IE.PRI	Privilege violation
IE.EOF	End of file
IE.ABO	Request terminated
IE.VER	Check sum error (This code occurs only if the device is treated as a sequential file device.)

13 Cassette Handler

13.1 Introduction

The cassette handler supports the TA11 magnetic tape cassette (a TA11 controller with a TU60 dual transport). Programming for cassette is quite similar to programming for magnetic tape. The TA11 system is a dual-drive, reel-to-reel unit which uses Philips-type cassettes.

The maximum capacity of a cassette, in bytes, is 92,000 (minus 300 per file gap and 40 per interrecord gap). It can transfer data at speeds of up to 562 bytes per second. Recording density is from 350 to 700 bits per inch, depending on tape position.

13.2 QIO MACRO

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

13.2.1 Standard QIO Functions

Table 13–1 lists the standard functions of the QIO macro that are valid for the tape cassette handler.

Table 13–1 Standard QIO Functions for the Tape Cassette Handler

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).

13.2.2 Device-Specific QIO Functions

Table 13–2 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 13–2 Device-Specific QIO Functions for the Tape Cassette Handler

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).

13.3 Status Returns

The error and status conditions listed in Table 13–3 are returned by the cassette handler described in this chapter.

Table 13–3 Tape Cassette Handler 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.
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.
IE.DAO	Data overrun.
	The handler 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 that the unit is off line.
IE.EOF	End-of-file encountered

Table 13–3 (Cont.) Tape Cassette Handler Status Returns

Code	Reason
	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 encountered EOT before the last block was completely written. It is recommended that the user rewrite the block, in its entirety, on another cassette.
IE.IFC	Illegal function
	A function code was specified in an I/O request that is illegal for cassette.
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	Unrecoverable error
	This code is returned when a block check error occurs. 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. If an unrecoverable 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.

13.3.1 Cassette Error 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 ensures 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.

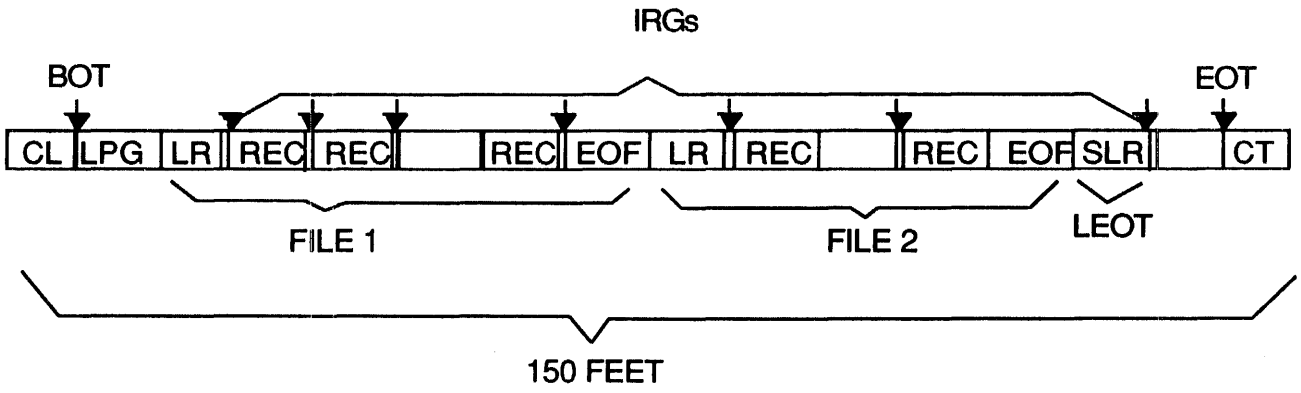
13.4 Structure of Cassette Tape

Figure 13–1 illustrates a general structure for cassette tape. A different structure can be employed if the user desires to do so.

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 13–1 One Possible 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

13.5 Programming Information

This section contains important programming considerations of which users of the cassette handler described in this chapter should be aware.

13.5.1 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).

13.5.2 End-of-File and IO.SPB

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 (that is, 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.

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

IO.SPB always stops in an IRG, IO.SPF in an EOF gap. Neither space function actually takes effect until data are 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.

13.5.4 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.

13.5.5 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 to occur.

13.5.6 Logical End-of-Tape

The conventional method of signalling 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.

14 Null Device Handler

14.1 Introduction

IAS provides the facility for input from and output to a "null device". QIOs to the null device have the following results:

QIO	I/O Status Returned
Read functions	IE.EOF
Write functions	IS.SUC
Attach	IS.SUC (if successfully attached)
	IE.DAA (if already attached)
Detach	IS.SUC (if successfully detached)
	IE.DNA (if not attached)
All others	IS.SUC

The null device is particularly useful for program testing. A program which is written to do I/O to a real device can temporarily be assigned to the null device (pseudo device NL:) while other parts of the program are being tested.

14.2 Example

Consider a program TESTPROG which, when tested and run with live data, produces:

- a listing report on lineprinter using LUN 7
- an output data file on magnetic tape using LUN 8

For test purposes the program has debugging dialogue which uses LUN 9 to communicate with the terminal, TI.

During initial testing the listing report and output data file are not required and the debugging dialogue only is used:

```
ASSIGN  NL: 7
ASSIGN  NL: 8
ASSIGN  TI: 9
RUN     TESTPROG
```

During final testing the output files are being checked and the debugging dialogue is disabled:

```
ASSIGN  LP0: 7
ASSIGN  MT1: 8
ASSIGN  NL:  9
RUN     TESTPROG
```

14.3 Prerequisites

Before the null device facility can be used, the null device handler (NL....) must be installed and loaded and the pseudo device NL: must have been defined during system generation.

15 DECtape II Handler

15.1 Introduction

The DECTAPE II (TU58) driver supports TU58 system hardware, providing low-cost, block-replaceable mass storage.

15.1.1 TU58 Hardware

Each TU58 DECTAPE II system consists of one or two TU58 cartridge drives, one tape drive controller, and one DL11-type serial line interface. Each TU58 drive functions as a random access, block-formatted mass storage device. Each tape cartridge is capable of storing 512 blocks of 512 bytes each. Access time is 10 seconds, average. All I/O transfers (commands and data) are via the serial line interface at serial transmission rates of 9600 baud. All read and write check operations are performed by the controller hardware using a 16-bit checksum. The controller performs up to 8 attempts to read a block, as necessary, before aborting the read operation and returning a hard error; however, whenever more than one read attempt is required for a successful read, the handler is notified in order to report a soft error message to the error logger.

15.1.2 TU58 Handler

The TU58 handler communicates with the TU58 hardware via a serial line interface (DL11); no other interface is required. All data and command transfers between the PDP-11 system and the TU58 are via programmed I/O and interrupt-driven routines; non-processor (NPR) data transfers are not supported.

The TU58 handler is installed with DD.... as the task name.

15.2 QIO MACRO

This section summarizes standard and device-specific QIO functions for the TU58.

15.2.1 Standard QIO Functions

Table 15–1 lists the standard QIO system directive functions of the QIO macro that are valid for the TU58.

Table 15–1 Standard QIO Functions for the TU58

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,,,lbn>	Read logical block

Table 15–1 (Cont.) Standard QIO Functions for the TU58

Format	Function
QIO$C IO.WLB,...,<stadd,size,,,lbn>	Write logical block

* I/O operations that are in progress when IO.KIL is received are allowed to complete. I/O requests that are queued when IO.KILL is received are killed.

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 cartridge tape where the data transfer starts (must be in the range of 0-777)

15.2.2 Device-Specific QIO Functions

The device-specific QIO functions for the TU58 are summarized in Table 15–2 and described in the following sections.

Table 15–2 Device-Specific QIO Functions for the TU58

Format	Function
QIO$C IO.WLC,...,<stadd,size,,,lbn>	Write logical block with check
QIO$C IO.RLC,...,<stadd,size,,,lbn>	Read logical block with check
QIO$C IO.BLS,...,<lbn>	Position tape
QIO$C IO.DGN,...	Run internal diagnostics

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 cartridge tape where the data transfer starts (must be in the range of 0-777)

IO.WLC

The IO.WLC function writes the specified data onto the tape cartridge. A checksum verification is then performed by reading the data just written; data is not returned to the task issuing the function. An appropriate status, based on the checksum verification, is returned to the issuing task.

IO.RLC

The IO.RLC function reads the tape with an increased threshold in the TU58's data recovery circuit. This is done as a check to insure data read reliability.

IO.BLS

The IO.BLS function is used for diagnostic purposes to position the tape to the specified logical block number.

IO.DGN

The IO.DGN function is used for diagnostic purposes to execute the TU58's internal (firmware) diagnostics. Appropriate status information is returned to the issuing task via the I/O status block.

15.3 Status Returns

Table 15–3 lists the error and status conditions that are returned by the TU58 handler.

Table 15–3 TU58 Handler 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.
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 the TU58.
IE.FHE	Fatal hardware error
	The motor has stopped.
IE.TMO	Timeout error
	The TU58 failed to respond to a function within the normal time specified by the handler.
IE.VER	Unrecoverable error
	The controller made its standard number of retries (8) after an error, but could not complete the operation successfully.
IE.WLK	Cartridge write-locked
	The task attempted to write on a tape cartridge that is physically write-locked.

15.4 Characteristics Words for DECTAPE II

The format of characteristics words is the same as for disc devices. Chapter 4, Section 4.6 describes the format fully.

A Listing of QIOMAC

```
 1            .TITLE  QIOMAC - QIOSYM MACRO DEFINITION
 2    ;
 3    ; DATE OF LAST MODIFICATION:
 4    ;
 5    ;       J.A. KASSON     5-FEB-80
 6    ;
 7    ;
 8    ; ***** ALWAYS UPDATE THE FOLLOWING TWO LINES TOGETHER
 9            .IDENT  /0340/
10            QI.VER=0340
11
12    ;
13    ; COPYRIGHT (C) 1980
14    ; DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
15    ;
16    ; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A
17    ; SINGLE COMPUTER SYSTEM AND MAY  BE  COPIED   ONLY  WITH  THE
18    ; INCLUSION OF THE ABOVE COPYRIGHT NOTICE.  THIS SOFTWARE,  OR
19    ; ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED  OR  OTHERWISE
20    ; MADE AVAILABLE TO ANY OTHER PERSON   EXCEPT FOR  USE ON SUCH
21    ; SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE  TERMS.  TITLE
22    ; TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL  TIMES  REMAIN
23    ; IN DEC.
24    ;
25    ; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
26    ; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
27    ; EQUIPMENT CORPORATION.
28    ;
29    ; DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
30    ; ITS SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.
31    ;
32    ;
33    ; PETER H. LIPMAN 1-OCT-73
34    ;
35    ;+
36    ; MACRO TO DEFINE STANDARD QUEUE I/O DIRECTIVE FUNCTION VALUES
37    ; AND IOSB RETURN VALUES.  TO INVOKE AT ASSEMBLY TIME (WITH LOCAL
38    ; DEFINITION) USE:
39    ;
40    ;       QIOSY$          ;DEFINE SYMBOLS
41    ;
42    ; TO OBTAIN GLOBAL DEFINITION OF THESE SYMBOLS USE:
43    ;
44    ;       QIOSY$ DEF$G    ;SYMBOLS DEFINED GLOBALLY
45    ;
46    ; THE MACRO CAN BE CALLED ONCE ONLY AND THEN
47    ; REDEFINES ITSELF AS NULL.
48    ;-
49
50            .MACRO  QIOSY$  $$$GBL,$$$MSG
51            .IIF    IDN,<$$$GBL>,<DEF$G>,   .GLOBL  QI.VER
52            .IF     IDN,<$$$MSG>,<DEF$S>
53            $$$MAX=0
54            $$MSG=1
55            .IFF
```

```
56            $$MSG=0
57            .ENDC
58            .MCALL  IOERR$
59            IOERR$  $$$GBL          ;I/O ERROR CODES FROM HANDLERS, FCP, FCS
60            .MCALL  DRERR$
61            DRERR$  $$$GBL          ;DIRECTIVE STATUS WORD ERROR CODES
62            .IF     DIF,<$$$MSG>,<DEF$S>
63            .MCALL  FILIO$
64            FILIO$  $$$GBL          ;DEFINE GENERAL I/O FUNCTION CODES
65            .MCALL  SPCIO$
66            SPCIO$  $$$GBL          ;DEVICE DEPENDENT I/O FUNCTION CODES
67            .MACRO  QIOSY$  ARG,ARG1,ARG2   ;RECLAIM MACRO STORAGE
68            .ENDM   QIOSY$
69            .ENDC
70            .ENDM   QIOSY$
71
72
73      ;
74      ; DEFINE THE ERROR CODES RETURNED BY DEVICE HANDLER AND FILE PRIMITIVES
75      ; IN THE FIRST WORD OF THE I/O STATUS BLOCK
76      ; THESE CODES ARE ALSO RETURNED BY FILE CONTROL SERVICES (FCS) IN THE
77      ; BYTE F.ERR IN THE FILE DESCRIPTOR BLOCK (FDB)
78      ;       THE BYTE F.ERR+1 IS 0 IF F.ERR CONTAINS A HANDLER OR FCP ERROR COI
79      ;
80            .MACRO  IOERR$  $$$GBL
81            .MCALL  .IOER.,DEFIN$
82            .IF     IDN,<$$$GBL>,<DEF$G>
83            ...GBL=1
84            .IFF
85            ...GBL=0
86            .ENDC
87            .IIF    NDF,$$MSG,$$MSG=0
88
89
90      ;
91      ; SYSTEM STANDARD CODES, USED BY EXECUTIVE AND DRIVERS
92      ;
93
94            .IOER.  IE.BAD,-01.,<BAD PARAMETERS>
95            .IOER.  IE.IFC,-02.,<INVALID FUNCTION CODE>
96            .IOER.  IE.DNR,-03.,<DEVICE NOT READY>
97            .IOER.  IE.VER,-04.,<PARITY ERROR ON DEVICE>
98            .IOER.  IE.ONP,-05.,<HARDWARE OPTION NOT PRESENT>
99            .IOER.  IE.SPC,-06.,<ILLEGAL USER BUFFER>
100           .IOER.  IE.DNA,-07.,<DEVICE NOT ATTACHED>
101           .IOER.  IE.DAA,-08.,<DEVICE ALREADY ATTACHED>
102           .IOER.  IE.DUN,-09.,<DEVICE NOT ATTACHABLE>
103           .IOER.  IE.EOF,-10.,<END OF FILE DETECTED>
104           .IOER.  IE.EOV,-11.,<END OF VOLUME DETECTED>
105           .IOER.  IE.WLK,-12.,<WRITE ATTEMPTED TO LOCKED UNIT>
106           .IOER.  IE.DAO,-13.,<DATA OVERRUN>
107           .IOER.  IE.SRE,-14.,<SEND/RECEIVE FAILURE>
108           .IOER.  IE.ABO,-15.,<REQUEST TERMINATED>
109           .IOER.  IE.PRI,-16.,<PRIVILEGE VIOLATION>
110           .IOER.  IE.RSU,-17.,<SHARABLE RESOURCE IN USE>
111           .IOER.  IE.OVR,-18.,<ILLEGAL OVERLAY REQUEST>
112           .IOER.  IE.BYT,-19.,<ODD BYTE COUNT (OR VIRTUAL ADDRESS)>
113           .IOER.  IE.BLK,-20.,<LOGICAL BLOCK NUMBER TOO LARGE>
114           .IOER.  IE.MOD,-21.,<INVALID UDC MODULE #>
115           .IOER.  IE.CON,-22.,<UDC CONNECT ERROR>
116           .IOER.  IE.BBE,-56.,<BAD BLOCK ON DEVICE>
117           .IOER.  IE.STK,-58.,<NOT ENOUGH STACK SPACE (FCS OR FCP)>
118           .IOER.  IE.FHE,-59.,<FATAL HARDWARE ERROR ON DEVICE>
```

```
119             .IOER.  IE.EOT,-62.,<END OF TAPE DETECTED>
120             .IOER.  IE.OFL,-65.,<DEVICE OFF LINE>
121             .IOER.  IE.BCC,-66.,<BLOCK CHECK, CRC, OR FRAMING ERROR>
122
123
124     ;
125     ; FILE PRIMITIVE CODES
126     ;
127
128             .IOER.  IE.NOD,-23.,<CALLER'S NODES EXHAUSTED>
129             .IOER.  IE.DFU,-24.,<DEVICE FULL>
130             .IOER.  IE.IFU,-25.,<INDEX FILE FULL>
131             .IOER.  IE.NSF,-26.,<NO SUCH FILE>
132             .IOER.  IE.LCK,-27.,<LOCKED FROM READ/WRITE ACCESS>
133             .IOER.  IE.HFU,-28.,<FILE HEADER FULL>
134             .IOER.  IE.WAC,-29.,<ACCESSED FOR WRITE>
135             .IOER.  IE.CKS,-30.,<FILE HEADER CHECKSUM FAILURE>
136             .IOER.  IE.WAT,-31.,<ATTRIBUTE CONTROL LIST FORMAT ERROR>
137             .IOER.  IE.RER,-32.,<FILE PROCESSOR DEVICE READ ERROR>
138             .IOER.  IE.WER,-33.,<FILE PROCESSOR DEVICE WRITE ERROR>
139             .IOER.  IE.ALN,-34.,<FILE ALREADY ACCESSED ON LUN>
140             .IOER.  IE.SNC,-35.,<FILE ID, FILE NUMBER CHECK>
141             .IOER.  IE.SQC,-36.,<FILE ID, SEQUENCE NUMBER CHECK>
142             .IOER.  IE.NLN,-37.,<NO FILE ACCESSED ON LUN>
143             .IOER.  IE.CLO,-38.,<FILE WAS NOT PROPERLY CLOSED>
144             .IOER.  IE.DUP,-57.,<ENTER - DUPLICATE ENTRY IN DIRECTORY>
145             .IOER.  IE.BVR,-63.,<BAD VERSION NUMBER>
146             .IOER.  IE.BHD,-64.,<BAD FILE HEADER>
147             .IOER.  IE.EXP,-75.,<FILE EXPIRATION DATE NOT REACHED>
148             .IOER.  IE.BTF,-76.,<BAD TAPE FORMAT>
149             .IOER.  IE.ALC,-84.,<ALLOCATION FAILURE>
150             .IOER.  IE.ULK,-85.,<UNLOCK ERROR>
151             .IOER.  IE.WCK,-86.,<WRITE CHECK FAILURE>
152             .IOER.  IE.DSQ,-90.,<DISK QUOTA EXCEEDED>
153
154     ;
155     ; FILE CONTROL SERVICES CODES
156     ;
157
158             .IOER.  IE.NBF,-39.,<OPEN - NO BUFFER SPACE AVAILABLE FOR FILE>
159             .IOER.  IE.RBG,-40.,<ILLEGAL RECORD SIZE>
160             .IOER.  IE.NBK,-41.,<FILE EXCEEDS SPACE ALLOCATED, NO BLOCKS>
161             .IOER.  IE.ILL,-42.,<ILLEGAL OPERATION ON FILE DESCRIPTOR BLOCK>
162             .IOER.  IE.BTP,-43.,<BAD RECORD TYPE>
163             .IOER.  IE.RAC,-44.,<ILLEGAL RECORD ACCESS BITS SET>
164             .IOER.  IE.RAT,-45.,<ILLEGAL RECORD ATTRIBUTES BITS SET>
165             .IOER.  IE.RCN,-46.,<ILLEGAL RECORD NUMBER - TOO LARGE>
166             .IOER.  IE.2DV,-48.,<RENAME - 2 DIFFERENT DEVICES>
167             .IOER.  IE.FEX,-49.,<RENAME - NEW FILE NAME ALREADY IN USE>
168             .IOER.  IE.BDR,-50.,<BAD DIRECTORY FILE>
169             .IOER.  IE.RNM,-51.,<CAN'T RENAME OLD FILE SYSTEM>
170             .IOER.  IE.BDI,-52.,<BAD DIRECTORY SYNTAX>
171             .IOER.  IE.FOP,-53.,<FILE ALREADY OPEN>
172             .IOER.  IE.BNM,-54.,<BAD FILE NAME>
173             .IOER.  IE.BDV,-55.,<BAD DEVICE NAME>
174             .IOER.  IE.NFI,-60.,<FILE ID WAS NOT SPECIFIED>
175             .IOER.  IE.ISQ,-61.,<ILLEGAL SEQUENTIAL OPERATION>
176             .IOER.  IE.NNC,-77.,<NOT ANSI "D" FORMAT BYTE COUNT>
177
178     ;
179     ; NETWORK ACP CODES
180     ;
181
```

```
182             .IOER.  IE.AST,-80.,<NO AST SPECIFIED IN CONNECT>
183             .IOER.  IE.NNN,-68.,<NO SUCH NODE>
184             .IOER.  IE.NFW,-69.,<PATH LOST TO PARTNER> ;THIS CODE MUST BE ODD
185             .IOER.  IE.BLB,-70.,<BAD LOGICAL BUFFER>
186             .IOER.  IE.TMM,-71.,<TOO MANY OUTSTANDING MESSAGES>
187             .IOER.  IE.NDR,-72.,<NO DYNAMIC SPACE AVAILABLE>
188             .IOER.  IE.CNR,-73.,<CONNECTION REJECTED>
189             .IOER.  IE.TMO,-74.,<TIMEOUT ON REQUEST>
190             .IOER.  IE.NNL,-78.,<NOT A NETWORK LUN>
191
192     ;
193     ; ICS/ICR ERROR CODES
194     ;
195             .IOER.  IE.NLK,-79.,<TASK NOT LINKED TO SPECIFIED ICS/ICR INTERRU
196             .IOER.  IE.NST,-80.,<SPECIFIED TASK NOT INSTALLED>
197             .IOER.  IE.FLN,-81.,<DEVICE OFFLINE WHEN OFFLINE REQUEST WAS ISSU
198
199
200     ;
201     ; TTY ERROR CODES
202     ;
203
204             .IOER.  IE.IES,-82.,<INVALID ESCAPE SEQUENCE>
205             .IOER.  IE.PES,-83.,<PARTIAL ESCAPE SEQUENCE>
206
207
208     ;
209     ; RECONFIGURATION CODES
210     ;
211
212             .IOER.  IE.ICE,-47.,<INTERNAL CONSISTANCY ERROR>
213             .IOER.  IE.ONL,-67.,<DEVICE ONLINE>
214
215     ;
216     ; PCL ERROR CODES
217     ;
218
219             .IOER.  IE.NTR,-87.,<TASK NOT TRIGGERED>
220             .IOER.  IE.REJ,-88.,<TRANSFER REJECTED BY RECEIVING CPU>
221             .IOER.  IE.FLG,-89.,<EVENT FLAG ALREADY SPECIFIED>
222
223
224     ;
225     ; SUCCESSFUL RETURN CODES---
226     ;
227
228             DEFIN$  IS.PND,+00.     ;OPERATION PENDING
229             DEFIN$  IS.SUC,+01.     ;OPERATION COMPLETE, SUCCESS
230             DEFIN$  IS.RDD,+02.     ;FLOPPY DISK SUCCESSFUL COMPLETION
231                                     ;OF A READ PHYSICAL, AND DELETED
232                                     ;DATA MARK WAS SEEN IN SECTOR HEADER
233             DEFIN$  IS.TNC,+02.     ;(PCL) SUCCESSFUL TRANSFER BUT MESSAGE
234                                     ;TRUNCATED (RECEIVE BUFFER TOO SMALL).
235             DEFIN$  IS.BV,+05.      ;(A/D READ) AT LEAST ONE BAD VALUE
236                                     ;WAS READ (REMAINDER MAY BE GOOD).
237                                     ;BAD CHANNEL IS INDICATED BY A
238                                     ;NEGATIVE VALUE IN THE BUFFER.
239
240
241     ;
242     ; TTY SUCCESS CODES
243     ;
244
```

```
245             DEFIN$  IS.CR,<15*400+1>  ;CARRIAGE RETURN WAS TERMINATOR
246             DEFIN$  IS.ESC,<33*400+1> ;ESCAPE (ALTMODE) WAS TERMINATOR
247             DEFIN$  IS.CC,<3*400+1> ;CONTROL-C WAS TERMINATOR
248             DEFIN$  IS.ESQ,<233*400+1>  ;ESCAPE SEQUENCE WAS TERMINATOR
249             DEFIN$  IS.PES,<200*400+1>  ;PARTIAL ESCAPE SEQUENCE TERMINATOR
250             DEFIN$  IS.EOT,<4*400+1>  ;EOT WAS TERMINATOR (BLOCK MODE INPUT)
251             DEFIN$  IS.TAB,<11*400+1> ;TAB WAS TERMINATOR (FORMS MODE INPUT)
252             DEFIN$  IS.TMO,+2.      ;REQUEST TIMED OUT
253
254
255     ; ******
256     ;
257     ; THE NEXT AVAILABLE ERROR NUMBER IS: -90.
258     ; ALL LOWER NUMBERS ARE IN USE !!
259     ;
260     ; *****
261             .IF     EQ,$$MSG
262             .MACRO  IOERR$  A
263             .ENDM   IOERR$
264             .ENDC
265             .ENDM   IOERR$
266
267     ;
268     ; DEFINE THE DIRECTIVE ERROR CODES RETURNED IN THE DIRECTIVE STATUS WORD
269     ;
270     ;       FILE CONTROL SERVICES (FCS) RETURNS THESE CODES IN THE BYTE F.ERR
271     ; OF THE FILE DESCRIPTOR BLOCK (FDB).  TO DISTINGUISH THEM FROM THE
272     ; OVERLAPPING CODES FROM HANDLER AND FILE PRIMITIVES, THE BYTE
273     ; F.ERR+1 IN THE FDB WILL BE NEGATIVE FOR A DIRECTIVE ERROR CODE.
274     ;
275             .MACRO  DRERR$  $$$GBL
276             .MCALL  .QIOE.,DEFIN$
277             .IF     IDN,<$$$GBL>,<DEF$G>
278             ...GBL=1
279             .IFF
280             ...GBL=0
281             .ENDC
282             .IIF    NDF,$$MSG,$$MSG=0
283     ;
284     ; STANDARD ERROR CODES RETURNED BY DIRECTIVES IN THE DIRECTIVE STATUS WOR
285     ;
286             .QIOE.  IE.UPN,-01.,<INSUFFICIENT DYNAMIC STORAGE>
287             .QIOE.  IE.INS,-02.,<SPECIFIED TASK NOT INSTALLED>
288             .QIOE.  IE.PTS,-03.,<PARTITION TOO SMALL FOR TASK>
289             .QIOE.  IE.UNS,-04.,<INSUFFICIENT DYNAMIC STORAGE FOR SEND>
290             .QIOE.  IE.ULN,-05.,<UN-ASSIGNED LUN>
291             .QIOE.  IE.HWR,-06.,<DEVICE HANDLER NOT RESIDENT>
292             .QIOE.  IE.ACT,-07.,<TASK NOT ACTIVE>
293             .QIOE.  IE.ITS,-08.,<DIRECTIVE INCONSISTENT WITH TASK STATE>
294             .QIOE.  IE.FIX,-09.,<TASK ALREADY FIXED/UNFIXED>
295             .QIOE.  IE.CKP,-10.,<ISSUING TASK NOT CHECKPOINTABLE>
296             .QIOE.  IE.TCH,-11.,<TASK IS CHECKPOINTABLE>
297             .QIOE.  IE.RBS,-15.,<RECEIVE BUFFER IS TOO SMALL>
298             .QIOE.  IE.PRI,-16.,<PRIVILEGE VIOLATION>
299             .QIOE.  IE.RSU,-17.,<RESOURCE IN USE>
300             .QIOE.  IE.NSW,-18.,<NO SWAP SPACE AVAILABLE>
301             .QIOE.  IE.ILV,-19.,<ILLEGAL VECTOR SPECIFIED>
302     ;
303     ;
304     ;
305             .QIOE.  IE.AST,-80.,<DIRECTIVE ISSUED/NOT ISSUED FROM AST>
306             .QIOE.  IE.MAP,-81.,<ILLEGAL MAPPING SPECIFIED>
307             .QIOE.  IE.IOP,-83.,<WINDOW HAS I/O IN PROGRESS>
```

```
308             .QIOE.  IE.ALG,-84.,<ALIGNMENT ERROR>
309             .QIOE.  IE.WOV,-85.,<ADDRESS WINDOW ALLOCATION OVERFLOW>
310             .QIOE.  IE.NVR,-86.,<INVALID REGION ID>
311             .QIOE.  IE.NVW,-87.,<INVALID ADDRESS WINDOW ID>
312             .QIOE.  IE.ITP,-88.,<INVALID TI PARAMETER>
313             .QIOE.  IE.IBS,-89.,<INVALID SEND BUFFER SIZE ( .GT. 255.)>
314             .QIOE.  IE.LNL,-90.,<LUN LOCKED IN USE>
315             .QIOE.  IE.IUI,-91.,<INVALID UIC>
316             .QIOE.  IE.IDU,-92.,<INVALID DEVICE OR UNIT>
317             .QIOE.  IE.ITI,-93.,<INVALID TIME PARAMETERS>
318             .QIOE.  IE.PNS,-94.,<PARTITION/REGION NOT IN SYSTEM>
319             .QIOE.  IE.IPR,-95.,<INVALID PRIORITY ( .GT. 250.)>
320             .QIOE.  IE.ILU,-96.,<INVALID LUN>
321             .QIOE.  IE.IEF,-97.,<INVALID EVENT FLAG ( .GT. 64.)>
322             .QIOE.  IE.ADP,-98.,<PART OF DPB OUT OF USER'S SPACE>
323             .QIOE.  IE.SDP,-99.,<DIC OR DPB SIZE INVALID>
324     ;
325     ; SUCCESS CODES FROM DIRECTIVES - PLACED IN THE DIRECTIVE STATUS WORD
326     ;
327             DEFIN$  IS.CLR,0        ;EVENT FLAG WAS CLEAR
328                                     ;FROM CLEAR EVENT FLAG DIRECTIVE
329             DEFIN$  IS.SET,2        ;EVENT FLAG WAS SET
330                                     ;FROM SET EVENT FLAG DIRECTIVE
331             DEFIN$  IS.SPD,2        ;TASK WAS SUSPENDED
332     ;
333     ;
334             .IF     EQ,$$MSG
335             .MACRO  DRERR$  A
336             .ENDM   DRERR$
337             .ENDC
338             .ENDM   DRERR$
339
340     ;
341     ; DEFINE THE GENERAL I/O FUNCTION CODES - DEVICE INDEPENDENT
342     ;
343             .MACRO  FILIO$  $$$GBL
344             .MCALL  .WORD.,DEFIN$
345             .IF     IDN,<$$$GBL>,<DEF$G>
346             ...GBL=1
347             .IFF
348             ...GBL=0
349             .ENDC
350     ;
351     ; GENERAL I/O QUALIFIER BYTE DEFINITIONS
352     ;
353             .WORD.  IQ.X,001,000    ;NO ERROR RECOVERY
354             .WORD.  IQ.Q,002,000    ;QUEUE REQUEST IN EXPRESS QUEUE
355             .WORD.  IQ.S,004,000    ;SYNONYM FOR IQ.UMD
356             .WORD.  IQ.UMD,004,000  ;USER MODE DIAGNOSTIC STATUS REQUIRED
357     ;
358     ; EXPRESS QUEUE COMMANDS
359     ;
360
361             .WORD.  IO.KIL,012,000  ;KILL CURRENT REQUEST
362             .WORD.  IO.RDN,022,000  ;I/O RUNDOWN
363             .WORD.  IO.UNL,042,000  ;UNLOAD I/O HANDLER TASK
364             .WORD.  IO.LTK,050,000  ;LOAD A TASK IMAGE FILE
365             .WORD.  IO.RTK,060,000  ;RECORD A TASK IMAGE FILE
366             .WORD.  IO.SET,030,000  ;SET CHARACTERISTICS FUNCTION
367     ;
368     ; GENERAL DEVICE HANDLER CODES
369     ;
370             .WORD.  IO.WLB,000,001  ;WRITE LOGICAL BLOCK
```

```
371             .WORD.  IO.RLB,000,002  ;READ LOGICAL BLOCK
372             .WORD.  IO.LOV,010,002  ;LOAD OVERLAY (DISK DRIVER)
373             .WORD.  IO.ATT,000,003  ;ATTACH A DEVICE TO A TASK
374             .WORD.  IO.DET,000,004  ;DETACH A DEVICE FROM A TASK
375     ;
376     ; DIRECTORY PRIMITIVE CODES
377     ;
378             .WORD.  IO.FNA,000,011  ;FIND FILE NAME IN DIRECTORY
379             .WORD.  IO.RNA,000,013  ;REMOVE FILE NAME FROM DIRECTORY
380             .WORD.  IO.ENA,000,014  ;ENTER FILE NAME IN DIRECTORY
381     ;
382     ; FILE PRIMITIVE CODES
383     ;
384             .WORD.  IO.CLN,000,007  ;CLOSE OUT LUN
385             .WORD.  IO.ULK,000,012  ;UNLOCK BLOCK
386             .WORD.  IO.ACR,000,015  ;ACCESS FOR READ
387             .WORD.  IO.ACW,000,016  ;ACCESS FOR WRITE
388             .WORD.  IO.ACE,000,017  ;ACCESS FOR EXTEND
389             .WORD.  IO.DAC,000,020  ;DE-ACCESS FILE
390             .WORD.  IO.RVB,000,021  ;READ VIRITUAL BLOCK
391             .WORD.  IO.WVB,000,022  ;WRITE VIRITUAL BLOCK
392             .WORD.  IO.EXT,000,023  ;EXTEND FILE
393             .WORD.  IO.CRE,000,024  ;CREATE FILE
394             .WORD.  IO.DEL,000,025  ;DELETE FILE
395             .WORD.  IO.RAT,000,026  ;READ FILE ATTRIBUTES
396             .WORD.  IO.WAT,000,027  ;WRITE FILE ATTRIBUTES
397             .WORD.  IO.APV,010,030  ;PRIVILEGED ACP CONTROL
398             .WORD.  IO.APC,000,030  ;ACP CONTROL
399     ;
400     ;
401             .MACRO  FILIO$  A
402             .ENDM   FILIO$
403             .ENDM   FILIO$
404
405     ;
406     ; DEFINE THE I/O FUNCTION CODES THAT ARE SPECIFIC TO INDIVIDUAL DEVICES
407     ;
408             .MACRO  SPCIO$  $$$GBL
409             .MCALL  .WORD.,DEFIN$
410             .IF     IDN,<$$$GBL>,<DEF$G>
411             ...GBL=1
412             .IFF
413             ...GBL=0
414             .ENDC
415     ;
416     ; I/O FUNCTION CODES FOR SPECIFIC DEVICE DEPENDENT FUNCTIONS
417     ;
418             .WORD.  IO.WLV,100,001  ;(DECTAPE) WRITE LOGICAL REVERSE
419             .WORD.  IO.WLS,010,001  ;(COMM.) WRITE PRECEDED BY SYNC TRAIN
420             .WORD.  IO.WNS,020,001  ;(COMM.) WRITE, NO SYNC TRAIN
421             .WORD.  IO.WAL,010,001  ;(TTY) WRITE PASSING ALL CHARACTERS
422             .WORD.  IO.WMS,020,001  ;(TTY) WRITE SUPPRESSIBLE MESSAGE
423             .WORD.  IO.CCO,040,001  ;(TTY) WRITE WITH CANCEL CONTROL-O
424             .WORD.  IO.WBT,100,001  ;(TTY) WRITE WITH BREAKTHROUGH
425             .WORD.  IO.WLT,010,001  ;(DISK) WRITE LAST TRACK
426             .WORD.  IO.WLC,020,001  ;(DISK) WRITE LOGICAL W/ WRITECHECK
427             .WORD.  IO.WPB,040,001  ;(DISK) WRITE PHYSICAL BLOCK
428             .WORD.  IO.WDD,140,001  ;(FLOPPY DISK) WRITE PHYSICAL W/ DELETED
429             .WORD.  IO.RLV,100,002  ;(MAGTAPE,DECTAPE) READ REVERSE
430             .WORD.  IO.RST,001,002  ;(TTY) READ WITH SPECIAL TERMINATOR
431             .WORD.  IO.RAL,010,002  ;(TTY) READ PASSING ALL CHARACTERS
432             .WORD.  IO.RNE,020,002  ;(TTY) READ WITHOUT ECHO
433             .WORD.  IO.RNC,040,002  ;(TTY) READ - NO LOWER CASE CONVERT
```

```
434             .WORD.  IO.RTM,200,002  ;(TTY) READ WITH TIME OUT
435             .WORD.  IO.RDB,200,002  ;(CARD READER) READ BINARY MODE
436             .WORD.  IO.SCF,200,002  ;(DISK) SHADOW COPY FUNCTION
437             .WORD.  IO.RHD,010,002  ;(COMM.) READ, STRIP SYNC
438             .WORD.  IO.RNS,020,002  ;(COMM.) READ, DON'T STRIP SYNC
439             .WORD.  IO.CRC,040,002  ;(COMM.) READ, DON'T CLEAR CRC
440             .WORD.  IO.RPB,040,002  ;(DISK) READ PHYSICAL BLOCK
441             .WORD.  IO.RLC,020,002  ;(DISK,MAGTAPE) READ LOGICAL W/ READCHECK
442             .WORD.  IO.ATA,010,003  ;(TTY) ATTACH WITH AST'S
443             .WORD.  IO.GTS,000,005  ;(TTY) GET TERMINAL SUPPORT CHARACTERISTI
444             .WORD.  IO.R1C,000,005  ;(AFC,AD01,UDC) READ SINGLE CHANNEL
445             .WORD.  IO.INL,000,005  ;(COMM.) INITIALIZATION FUNCTION
446             .WORD.  IO.TRM,010,005  ;(COMM.) TERMINATION FUNCTION
447             .WORD.  IO.RWD,000,005  ;(MAGTAPE,DECTAPE) REWIND
448             .WORD.  IO.SPB,020,005  ;(MAGTAPE) SPACE "N" BLOCKS
449             .WORD.  IO.SPF,040,005  ;(MAGTAPE) SPACE "N" EOF MARKS
450             .WORD.  IO.STC,100,005  ;SET CHARACTERISTIC
451             .WORD.  IO.SMD,110,005  ;(FLOPPY DISK) SET MEDIA DENSITY
452             .WORD.  IO.SEC,120,005  ;SENSE CHARACTERISTIC
453             .WORD.  IO.RWU,140,005  ;(MAGTAPE,DECTAPE) REWIND AND UNLOAD
454             .WORD.  IO.SMO,160,005  ;(MAGTAPE) MOUNT & SET CHARACTERISTICS
455             .WORD.  IO.HNG,000,006  ;(TTY) HANGUP DIAL-UP LINE
456             .WORD.  IO.RBC,000,006  ;READ MULTICHANNELS (BUFFER DEFINES CHANN
457             .WORD.  IO.MOD,000,006  ;(COMM.) SETMODE FUNCTION FAMILY
458             .WORD.  IO.HDX,010,006  ;(COMM.) SET UNIT HALF DUPLEX
459             .WORD.  IO.FDX,020,006  ;(COMM.) SET UNIT FULL DUPLEX
460             .WORD.  IO.SYN,040,006  ;(COMM.) SPECIFY SYNC CHARACTER
461             .WORD.  IO.EOF,000,006  ;(MAGTAPE) WRITE EOF
462             .WORD.  IO.ERS,020,006  ;(MAGTAPE) ERASE TAPE
463             .WORD.  IO.DSE,040,006  ;(MAGTAPE) DATA SECURITY ERASE
464             .WORD.  IO.RTC,000,007  ;READ CHANNEL - TIME BASED
465             .WORD.  IO.SAO,000,010  ;(UDC) SINGLE CHANNEL ANALOG OUTPUT
466             .WORD.  IO.SSO,000,011  ;(UDC) SINGLE SHOT, SINGLE POINT
467             .WORD.  IO.RPR,000,011  ;(TTY) READ WITH PROMPT
468             .WORD.  IO.MSO,000,012  ;(UDC) SINGLE SHOT, MULTI-POINT
469             .WORD.  IO.RTT,001,012  ;(TTY) READ WITH TERMINATOR TABLE
470             .WORD.  IO.SLO,000,013  ;(UDC) LATCHING, SINGLE POINT
471             .WORD.  IO.MLO,000,014  ;(UDC) LATCHING, MULTI-POINT
472             .WORD.  IO.LED,000,024  ;(LPS11) WRITE LED DISPLAY LIGHTS
473             .WORD.  IO.SDO,000,025  ;(LPS11) WRITE DIGITAL OUTPUT REGISTER
474             .WORD.  IO.SDI,000,026  ;(LPS11) READ DIGITAL INPUT REGISTER
475             .WORD.  IO.SCS,000,026  ;(UDC) CONTACT SENSE, SINGLE POINT
476             .WORD.  IO.REL,000,027  ;(LPS11) WRITE RELAY
477             .WORD.  IO.MCS,000,027  ;(UDC) CONTACT SENSE, MULTI-POINT
478             .WORD.  IO.ADS,000,030  ;(LPS11) SYNCHRONOUS A/D SAMPLING
479             .WORD.  IO.CCI,000,030  ;(UDC) CONTACT INT - CONNECT
480             .WORD.  IO.LOD,000,030  ;(LPA11) LOAD MICROCODE
481             .WORD.  IO.MDI,000,031  ;(LPS11) SYNCHRONOUS DIGITAL INPUT
482             .WORD.  IO.DCI,000,031  ;(UDC) CONTACT INT - DISCONNECT
483             .WORD.  IO.XMT,000,031  ;(COMM.) TRANSMIT SPECIFIED BLOCK WITH AC
484             .WORD.  IO.XNA,010,031  ;(COMM.) TRANSMIT WITHOUT ACK
485             .WORD.  IO.INI,000,031  ;(LPA11) INITIALIZE
486             .WORD.  IO.HIS,000,032  ;(LPS11) SYNCHRONOUS HISTOGRAM SAMPLING
487             .WORD.  IO.RCI,000,032  ;(UDC) CONTACT INT - READ
488             .WORD.  IO.RCV,000,032  ;(COMM.) RECEIVE DATA IN BUFFER SPECIFIED
489             .WORD.  IO.CLK,000,032  ;(LPA11) START CLOCK
490             .WORD.  IO.CSR,000,032  ;(BUS SWITCH) READ CSR REGISTER
491             .WORD.  IO.MDO,000,033  ;(LPS11) SYNCHRONOUS DIGITAL OUTPUT
492             .WORD.  IO.CTI,000,033  ;(UDC) TIMER - CONNECT
493             .WORD.  IO.CON,000,033  ;(COMM.) CONNECT FUNCTION
494                                     ;(VT11) - CONNECT TASK TO DISPLAY PROCESS
495                                     ;(BUS SWITCH) CONNECT TO SPECIFIED BUS
496             .WORD.  IO.STA,000,033  ;(LPA11) START DATA TRANSFER
```

```
497             .WORD.  IO.DTI,000,034  ;(UDC) TIMER - DISCONNECT
498             .WORD.  IO.DIS,000,034  ;(COMM.) DISCONNECT FUNCTION
499                                     ;(VT11) - DISCONNECT TASK FROM DISPLAY PR(
500                                     ;(BUS SWITCH) SWITCHED BUS DISCONNECT
501             .WORD.  IO.MDA,000,034  ;(LPS11) SYNCHRONOUS D/A OUTPUT
502             .WORD.  IO.DPT,010,034  ;(BUS SWITCH) DISCONNECT TO SPECIF PORT N(
503             .WORD.  IO.RTI,000,035  ;(UDC) TIMER - READ
504             .WORD.  IO.CTL,000,035  ;(COMM.) NETWORK CONTROL FUNCTION
505             .WORD.  IO.STP,000,035  ;(LPS11,LPA11) STOP IN PROGRESS FUNCTION
506                                     ;(VT11) - STOP DISPLAY PROCESSOR
507             .WORD.  IO.SWI,000,035  ;(BUS SWITCH) SWITCH BUSSES
508             .WORD.  IO.CNT,000,036  ;(VT11) - CONTINUE DISPLAY PROCESSOR
509             .WORD.  IO.ITI,000,036  ;(UDC) TIMER - INITIALIZE
510
511
512     ;
513     ; COMMUNICATIONS FUNCTIONS
514     ;
515
516             .WORD.  IO.CPR,010,033  ;CONNECT NO TIMEOUTS
517             .WORD.  IO.CAS,020,033  ;CONNECT WITH AST
518             .WORD.  IO.CRJ,040,033  ;CONNECT REJECT
519             .WORD.  IO.CBO,110,033  ;BOOT CONNECT
520             .WORD.  IO.CTR,210,033  ;TRANSPARENT CONNECT
521             .WORD.  IO.GNI,010,035  ;GET NODE INFORMATION
522             .WORD.  IO.GLI,020,035  ;GET LINK INFORMATION
523             .WORD.  IO.GLC,030,035  ;GET LINK INFO CLEAR COUNTERS
524             .WORD.  IO.GRI,040,035  ;GET REMOTE NODE INFORMATION
525             .WORD.  IO.GRC,050,035  ;GET REMOTE NODE ERROR COUNTS
526             .WORD.  IO.GRN,060,035  ;GET REMOTE NODE NAME
527             .WORD.  IO.CSM,070,035  ;CHANGE SOLO MODE
528             .WORD.  IO.CIN,100,035  ;CHANGE CONNECTION INHIBIT
529             .WORD.  IO.SPW,110,035  ;SPECIFY NETWORK PASSWORD
530             .WORD.  IO.CPW,120,035  ;CHECK NETWORK PASSWORD.
531             .WORD.  IO.NLB,130,035  ;NSP LOOPBACK
532             .WORD.  IO.DLB,140,035  ;DDCMP LOOPBACK
533
534     ;
535     ; ICS/ICR I/O FUNCTIONS
536     ;
537             .WORD.  IO.CTY,000,007  ;CONNECT TO TERMINAL INTERRUPTS
538             .WORD.  IO.DTY,000,015  ;DISCONNECT FROM TERMINAL INTERRUPTS
539             .WORD.  IO.LDI,000,016  ;LINK TO DIGITAL INTERRUPTS
540             .WORD.  IO.UDI,010,023  ;UNLINK FROM DIGITAL INTERRUPTS
541             .WORD.  IO.LTI,000,017  ;LINK TO COUNTER MODULE INTERRUPTS
542             .WORD.  IO.UTI,020,023  ;UNLINK FROM COUNTER MODULE INTERRUPTS
543             .WORD.  IO.LTY,000,020  ;LINK TO REMOTE TERMINAL INTERRUPTS
544             .WORD.  IO.UTY,030,023  ;UNLINK FROM REMOTE TERMINAL INTERRUPTS
545             .WORD.  IO.LKE,000,024  ;LINK TO ERROR INTERRUPTS
546             .WORD.  IO.UER,040,023  ;UNLINK FROM ERROR INTERRUPTS
547             .WORD.  IO.NLK,000,023  ;UNLINK FROM ALL INTERRUPTS
548             .WORD.  IO.ONL,000,037  ;UNIT ONLINE
549             .WORD.  IO.FLN,000,025  ;UNIT OFFLINE
550             .WORD.  IO.RAD,000,021  ;READ ACTIVATING DATA
551
552     ;
553     ; IP11 I/O FUNCTIONS
554     ;
555             .WORD.  IO.MAO,010,007  ;MULTIPLE ANALOG OUTPUTS
556             .WORD.  IO.LEI,010,017  ;LINK EVENT FLAGS TO INTERRUPT
557             .WORD.  IO.RDD,010,020  ;READ DIGITAL DATA
558             .WORD.  IO.RMT,020,020  ;READ MAPPING TABLE
559             .WORD.  IO.LSI,000,022  ;LINK TO DSI INTERRUPTS
```

```
560             .WORD.  IO.UEI,050,023  ;UNLINK EVENT FLAGS
561             .WORD.  IO.USI,060,023  ;UNLINK FROM DSI INTERRUPTS
562             .WORD.  IO.CSI,000,026  ;CONNECT TO DSI INTERRUPTS
563             .WORD.  IO.DSI,000,027  ;DISCONNECT FROM DSI INTERRUPTS
564
565     ;
566     ; PCL11 I/O FUNCTIONS
567     ;
568
569             .WORD.  IO.ATX,000,001  ;ATTEMPT TRANSMISSION
570             .WORD.  IO.ATF,000,002  ;ACCEPT TRANSFER
571             .WORD.  IO.CRX,000,031  ;CONNECT FOR RECEPTION
572             .WORD.  IO.DRX,000,032  ;DISCONNECT FROM RECEPTION
573             .WORD.  IO.RTF,000,033  ;REJECT TRANSFER
574
575
576             .MACRO  SPCIO$  A
577             .ENDM   SPCIO$
578             .ENDM   SPCIO$
579
580     ;
581     ; DEFINE THE I/O CODES FOR USER-MODE DIAGNOSITCS.  ALL DIAGNOSTIC
582     ; FUNCTION ARE IMPLEMENTED AS A SUBFUNCTION OF I/O CODE 10 (OCTAL).
583     ;
584
585             .MACRO  UMDIO$  $$$GBL
586             .MCALL  .WORD.,DEFIN$
587             .IF IDN <$$$GBL>,<DEF$G>
588     ...GBL=1
589             .IFF
590     ...GBL=0
591             .ENDC
592
593     ;
594     ; DEFINE THE GENERAL USER-MODE I/O QUALIFIER BIT.
595     ;
596
597             .WORD.  IQ.UMD,004,000  ;USER MODE DIAGNOSTIC REQUEST
598
599     ;
600     ; DEFINE USER-MODE DIAGNOSTIC FUNCTIONS.
601     ;
602
603             .WORD.  IO.HMS,000,010  ;(DISK) HOME SEEK OR RECALIBRATE
604             .WORD.  IO.BLS,010,010  ;(DISK) BLOCK SEEK
605             .WORD.  IO.OFF,020,010  ;(DISK) OFFSET POSITION
606             .WORD.  IO.RDH,030,010  ;(DISK) READ DISK HEADER
607             .WORD.  IO.WDH,040,010  ;(DISK) WRITE DISK HEADER
608             .WORD.  IO.WCK,050,010  ;(DISK) WRITECHECK (NON-TRANSFER)
609             .WORD.  IO.RNF,060,010  ;(DECTAPE) READ BLOCK NUMBER FORWARD
610             .WORD.  IO.RNR,070,010  ;(DECTAPE) READ BLOCK NUMBER REVERSE
611             .WORD.  IO.LPC,100,010  ;(MAGTAPE) READ LONGITUDINAL PARITY CHAR
612             .WORD.  IO.RTD,120,010  ;(DISK) READ TRACK DESCRIPTOR
613             .WORD.  IO.WTD,130,010  ;(DISK) WRITE TRACK DESCRIPTOR
614             .WORD.  IO.TDD,140,010  ;(DISK) WRITE TRACK DESCRIPTOR DISPLACED
615             .WORD.  IO.DGN,150,010  ;DIAGNOSE MICRO PROCESSOR FIRMWARE
616             .WORD.  IO.WPD,160,010  ;(DISK) WRITE PHYSICAL BLOCK
617             .WORD.  IO.RPD,170,010  ;(DISK) READ PHYSICAL BLOCK
618             .WORD.  IO.CER,200,010  ;(DISK) READ CE BLOCK
619             .WORD.  IO.CEW,210,010  ;(DISK) WRITE CE BLOCK
620
621     ;
622     ; MACRO REDEFINITION TO NULL
```

```
623     ;
624
625             .MACRO  UMDIO$  A
626             .ENDM
627
628
629             .ENDM   UMDIO$
630
631     ;
632     ; HANDLER ERROR CODES RETURNED IN I/O STATUS BLOCK ARE DEFINED THROUGH TH
633     ; MACRO WHICH THEN CONDITIONALLY INVOKES THE MESSAGE GENERATING MACRO
634     ; FOR THE QIOSYM.MSG FILE
635     ;
636             .MACRO  .IOER.  SYM,LO,MSG
637             DEFIN$  SYM,LO
638             .IF     GT,$$MSG
639             .MCALL  .IOMG.
640             .IOMG.  SYM,LO,<MSG>
641             .ENDC
642             .ENDM   .IOER.
643     ;
644     ; I/O ERROR CODES ARE DEFINED THOUGH THIS MACRO WHICH THEN INVOKES THE
645     ; ERROR MESSAGE GENERATING MACRO, ERROR CODES -129 THROUGH -256
646     ; ARE USED IN THE QIOSYM.MSG FILE
647     ;
648             .MACRO  .QIOE.  SYM,LO,MSG
649             DEFIN$  SYM,LO
650             .IF     GT,$$MSG
651             .MCALL  .IOMG.
652             .IOMG.  SYM,<LO-128.>,<MSG>
653             .ENDC
654             .ENDM   .QIOE.
655     ;
656     ; CONDITIONALLY GENERATE DATA FOR WRITING A MESSAGE FILE
657     ;
658             .MACRO  .IOMG.  SYM,LO,MSG
659             .WORD   -^O<LO>
660             .ASCIZ  ^MSG^
661             .EVEN
662             .IIF    LT,^O<$$$MAX+<LO>>,$$$MAX=-^O<LO>
663             .ENDM   .IOMG.
664     ;
665     ; DEFINE THE SYMBOL SYM WHERE LO IS IS THE LOW ORDER BYTE, HI IS THE HIGH
666     ;
667             .MACRO  .WORD.  SYM,LO,HI
668             DEFIN$  SYM,<HI*400+LO>
669             .ENDM   .WORD.
```

Index

A

C

D

E

F

I

K

L

M

N

P

Q

R

S

T

U

W

Reader's Comments

This form is for document comments only. Digital will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

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

Did you find errors in this manual? If so, specify the error and the page number.

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

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)_____

Name_____Date_____

Organization_____

Street_____

City_____State_____Zip Code_____
or Country

Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO 33 MAYNARD MASS

POSTAGE WILL BE PAID BY ADDRESSEE

IAS Engineering/Documentation
Digital Equipment Corporation
5 Wentworth Drive GSF/L20
Hudson, NH 03051-4929

Do Not Tear - Fold Here