IEX/VMS

User's and Installation Guide

Order Number: AA–X673D–TE

This document describes the IEX-VMS-Driver software for control of the IEU11-A/IEQ11-A module on the MicroVMS and VMS operating systems.

Prepared by U.S. Area EIC Documentation Services
Digital Equipment Corporation . Merrimack, NH 03054

4th Edition. July 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

TABLES

Preface

MANUAL OBJECTIVES

This manual describes the IEX-VMS-Driver. This software controls the IEX11 IEC/IEEE interface module in a MicroVMS or VMS operating system environment. The driver is accessed through the VMS system service routine, SYS$QIO, but this routine and system service routines in general are not described. The *VAX/VMS System Service Manual* covers this topic. The driver is also accessed through the IEX Simplified User Interface.

This manual also contains an overview and shows in programming examples how to handle certain IEC 625/IEEE-488 bus transactions.

INTENDED AUDIENCE

This manual is intended for all programmers writing code for the IEU11-A/IEQ11-A module. This manual assumes that the user is familiar with the IEC 625/IEEE-488 bus standards. It describes the driver's features and requirements. This manual assumes that the reader is familiar with programming terms and understands Digital addressing conventions and VAX architecture. In particular, the manual assumes that the user is already a competent high level language or MACRO programmer. The manual assumes that the user has some familiarity with the VMS I/O structure, particularly the use of the SYS$QIO system service routine. Some explanation is given of concepts such as AST routines and event flags, but a detailed description of these VMS features is not covered.

DOCUMENT STRUCTURE

The manual contains six chapters and four appendices. The following material is discussed:

Chapter	Title	Contents
Chapter 1	Introduction	A brief overview of the IEC 625/IEEE-488 bus standards and terminologies
Chapter 2	Installation	Templates of successful installations.
Chapter 3	Software Interface - I/O Functions	Descriptions of all the available function codes, code modifiers, arguments and descriptions of the type of information returned in the I/O status blocks.
Chapter 4	Simplified User Interface - Overview	Descriptions of subroutine calls accessible from higher level languages
Chapter 5	IEX Exerciser Program	Description of the Exerciser and all the available commands.
Chapter 6	Differences to IEC11-V Driver	Descriptions of the differences between the IEC11-V driver and the IEX-VMS-Driver, according to the corresponding interfaces IEC11-A/IEC11-B and IEU11-A/IEQ11-A.
Appendix A	Example of IEX Exerciser Using IEX$DEMO.COM	An example of the output of the IEX$DEMO.COM command file processed by the IEX Exerciser.
Appendix B	Example of Program Conversion from IEC11 to IEX11	Description of MACRO-32 application program differences between the IEC11-V driver and the IEX-VMS-Driver
Appendix C	IEC 625/IEEE-488 Bus Commands	Table of IEC 625/IEEE-488 bus commands and values
Appendix D	IEX-VMS-Driver V4.2 Kit Contents	Description of files contained on the IEX-VMS-Driver kit

ASSOCIATED DOCUMENTS

The following documents may be of interest to IEU11-A/IEQ11-A users.

- *IEU11-A/IEQ11-A User's Guide* for a layout of the IEU11-A/IEQ11-A device registers and a complete description of the auxiliary commands.
- *VAX/VMS System Services Reference Manual*
- *VAX/VMS I/O User's Reference Manual*
- *ANSI/IEEE Standard 488.1-1987* Specification for details on bus transactions.

CONVENTIONS USED IN THIS DOCUMENT

Industry standard mnemonics and acronyms for the IEEE 488-1 Interface are derived from the *IEEE Standard Digital Interface for Programmable Instrumentation*.

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

All addresses in this manual are assumed to be octal, and all numbers are assumed to be decimal. All other numbers are explicitly indicated and their notation depends on the programming language used. Examples:

Representation	Description
n_8	octal number n
n(hex)	hexadecimal number n
^On	octal number n (MACRO-32 syntax)
^Xn	hexadecimal number n (MACRO-32 syntax)
n.	decimal number n (IEX Exerciser syntax)
'n'X	hexadecimal number n (IEX Exerciser syntax)

For the IEU11-A/IEQ11-A and the IEC11-A/IEC11-B, the terms interface, module, device, and controller all have the same meaning and are used interchangeably.

The term IEX11 stands for either the IEU11-A or IEQ11-A controller. The term IEC11 stands for either the IEC11-A or IEC11-B controller.

The term IEC11-CA is an option name and includes the IEC11-A and IEC11-B controller.

1 OVERVIEW

1.1 OVERVIEW

The IEX-VMS-Driver allows programs written in MACRO-32 and higher level languages such as VAX FORTRAN, BASIC, C, and Pascal to communicate through the IEU11-A or IEQ11-A, with devices containing an IEC 625-1 or IEEE 488 interface. This communication is implemented through a choice of direct QIO calls or a set of subroutines callable from high level languages.

Note: In the following description, the term IEX11 stands for the IEU11-A and/or IEQ11-A interface. The IEU11-A supports the VAX UNIBUS processors and the IEQ11-A supports the MicroVAX Q-Bus processors.

The IEX11 interface is a bit-parallel byte-serial device which can perform transfers in either program interrupt or **direct memory access** (DMA) mode. Program interrupt is used for transferring commands and addresses, and DMA is used for transferring data. DMA significantly reduces processor loading during the transfer of data buffers.

The IEX11 interface contains two independent controllers for an IEC/IEEE bus, also known as **General Purpose Interface Buses** (GPIB). These two controllers may be connected to the same or to different IEC/IEEE buses. Up to 15 devices (including one or more IEX11 controllers) may be attached to each IEC/IEEE bus.

While the IEX11 is most commonly used as controller-in-charge on the bus, it can also be used with another device as controller-in-charge. The controller-in-charge directs the sequence of information on the bus.

When the IEX11 acts as controller-in-charge, it may request status information from other devices through serial and parallel polling. When the IEX11 is not controller-in-charge, it can be polled in either fashion to provide status information.

The following sections contain a brief description of:

- Lines on the IEC/IEEE Bus
- Controller Types and States
- Device Addressing
- Data Transferring
- Polling Methods
- Programming

1.2 IEC/IEEE BUS INTRODUCTION

The IEC/IEEE Bus is an international standard for an interface system to interconnect programmable instrumentation controllers. The specifications of the bus are described in the ANSI/IEEE Standard 488.1-1987 *IEEE Standard Digital Interface for Programmable Instrumentation*, published by the Institute of Electrical and Electronic Engineers. This document was produced in concert with the International Electrotechnical Commission. Interface systems conforming to these standards are referred to the United States as being IEEE or IEEE-488 compatible, while in Europe the abbreviation IEC or IEC 625-1 is used. In this context, the abbreviations IEEE and IEC represent standards which are technically and functionally identical.

1.2.1 IEC 625/IEEE-488 Bus

The IEC or IEEE standardized instrument bus allows the transfer of digital data within a group of instruments and system components. Information is transmitted in byte-serial, bit parallel format and may consist of either interface messages or device dependent messages. Device dependent messages and all seven bit interface messages are transmitted on a set of eight interface signal lines (data bus) in conjunction with a set of three interface signal lines (three-wire handshake). As a result of the handshake mechanism, the speed of data transmission is limited by the slowest device on the bus. The remaining five interface signal lines are used for management. Signals transferred through these lines are referred to as single line messages.

Devices which can be attached to the IEC/IEEE bus fall into four classes:

1 Devices able to talk only

2 Devices able to listen only

3 Devices able to talk and listen

4 Devices able to talk, listen and control

The simplest instrument is the one that can talk only. When addressed, this device transfers its data onto the bus in a predetermined format. Since such a device cannot receive instructions from the bus, its settings can only be altered through manual intervention.

Devices that listen only can read data from the data bus lines. In the case of a signal generator, the data could be instructions to the instrument to generate signals of the specified amplitude and frequency. Printers are frequently listen only instruments.

A digital multimeter is a device that can both listen and talk. The multimeter is configured by instructions from the device that is the controller on the bus. The multimeter then performs the requested readings and returns the results as data onto the bus.

The controller-in-charge controls the sequence of all operations on the IEC/IEEE bus.

The 16 lines of this bus form three functional groups:

- Interface Management lines - five lines
- Handshake (data-byte transfer) control lines - three lines
- Bidirectional Data Transfer lines - eight lines

Each of these groups are described in detail in the following sections.

1.2.2 Interface Management Lines

The five IEC/IEEE bus interface management lines are as follows:

1. Attention (ATN) specifies how data on the Data Input/Output (DIO) lines are to be interpreted and by which devices. ATN is asserted (low=1=TRUE) during transfers of IEC/IEEE commands and deasserted (high=0=FALSE) during data transfers.
2. Interface Clear (IFC) places the entire system into a quiescent state.
3. Service Request (SRQ) is used by a device to indicate the need for attention from the controller-in-charge and to request an interruption of the current sequence of events.
4. Remote Enable (REN) in conjunction with other messages selects alternate sources of device programming data (typically IEC/IEEE bus versus front panel).
5. End or Identify (EOI) indicates the end of a multiple byte data transfer.

When ATN is low (TRUE), the IEC/IEEE bus is in command mode. In command mode the controller is active and all other devices are waiting for instructions. Command mode instructions fall into five groups:

1. Talker Address Group (TAG) commands enable a device to talk. Only one device at a time may act as the talker. When the controller addresses one device to talk, the previous talker must be deaddressed and ceases to be a talker.
2. Listener Address Group (LAG) commands enable a specific device to listen. Several devices may be listeners at the same time.
3. Universal Command Group (UCG) commands can be responded to by devices on the IEC/IEEE bus regardless of their addressed states. However, some devices may not have the functionality to respond to UCG commands.
4. Addressed Command Group (ACG) commands are responded to only by devices that are addressed as listeners. As with UCG commands, some devices are not capable of responding to all ACG commands.
5. Secondary Command Group (SCG) commands are used when addressing extended listeners and talkers or enabling the parallel poll.

The unlisten address command (UNL) unaddresses all listeners that have been previously addressed to listen. The untalk address command (UNT) unaddresses any talker that had been previously addressed to talk.

1.2.3 Handshake Lines

The three IEC/IEEE bus handshake lines are as follows:

1. Data Valid (DAV) indicates the availability and validity of information on the DIO lines.
2. Not Ready For Data (NRFD) indicates the state of readiness of devices to accept data.
3. Not Data Accepted (NDAC) indicates the condition of acceptance of the data by device(s).

DAV, NRFD, and NDAC operate in a three-wire interlocked handshake process to transfer each data byte across the data lines.

A handshake sequence is entered with the listener-controlled NRFD and NDAC both low and DAV high. As each listener is ready to accept data, it releases its NRFD line. When all listeners have released the NRFD line, pull-up resistors on the line pull NRFD high. The talker signals new Data Valid by pulling the DAV line low. Listeners respond by putting their NRFD outputs low. During the period that listeners accept data, they release the NDAC line. When data has been accepted by all the listeners, the NDAC line goes high. Acknowledgment by the talker releases the DAV line and the handshake is completed by the listeners by pulling the NDAC line low. Note that the NRFD and NDAC lines may never go high (logic 0) together.

1.2.4 Data Transfer Lines

There are eight interface signal lines which serve for transferring all data, command, address, and status bytes. These bytes are carried on the data input/output signal (DIO<8:1>) in a bit-parallel, byte-serial form. Transfers are asynchronous and, generally, bidirectional.

1.3 SYSTEM CONTROLLER

Only one device on the IEC/IEEE bus can be designated as system controller. The role of the system controller is hardware oriented. The following two restrictions should be taken into account when programming the IEX11. According to the IEC/IEEE standards only the system controller may do the following:

1. Issue an interface clear (IFC) request, thus asserting the IFC line on the IEC/IEEE bus. The assertion of the interface clear resets all devices on the bus to a known quiescent state. It also makes the system controller the controller-in-charge. Normally, the system controller issues an interface clear request during initialization or after a severe error has been detected.
2. Issue a remote enable (REN) request, thus asserting the REN line on the IEC/IEEE bus. Asserting this line forces any device addressed as listener out of the local and into the remote state. When in the remote state, a device can be controlled by commands on the IEC/IEEE bus.

1.3.1 Controller-In-Charge

At any given time, only one device on the IEC/IEEE bus may be controller-in-charge (CIC). Initially, the system controller becomes controller-in-charge by issuing an interface clear. The role of controller-in-charge may be passed to any other device which has the necessary hardware to perform the control functions. Only the controller-in-charge has the privileges to do the following:

- Change the state of the attention (ATN) line. When this line is asserted, signals on the DIO lines represent commands to devices; when this line is reset, then the signals represent data.
- Send remote messages onto the IEC/IEEE bus. These messages are commands to other devices. The most common use of these commands is to change the address state of other devices.
- Issue a serial poll.
- Issue a parallel poll.

For the majority of applications, the same IEX11 unit is both the system controller and the controller-in-charge. An IEX11 unit is designated as system controller and/or controller-in-charge with an Initialize request (refer to the IEX exerciser commands INI and STS Section 5.2.10, and the $QIO function IO$_INITIALIZE Section 3.3.6).

In summary, all devices on the IEC/IEEE bus can send and receive data and receive commands if they have talker and listener capabilities, but only the controller-in-charge may issue IEC/IEEE bus commands.

1.3.2 Controller-State

The IEC/IEEE standards specify controller states and the conditions necessary for the transition to and from these states. Many of the states represent steps in the transition between major states and exist only for a matter of microseconds. Normally, you need not be concerned with the IEC/IEEE states, since the IEX-VMS-Driver will, whenever possible, enter the state required to perform the requested operation.

You should, however, be aware of the controller states monitored by the driver.

- Controller Active State (CACS)

  This is the state of the controller-in-charge when the ATN line is asserted. In this state the controller-in-charge can transmit commands across the IEC/IEEE bus.

- Controller Standby State (CSBS)

  This is the state of the controller-in-charge when the ATN line is deasserted. Only when the controller-in-charge is in this state can data be transferred across the IEC/IEEE bus.

- Controller Idle State (CIDS)

  This is the state of all devices which are not controller-in-charge, regardless of the state of the ATN line.

1.4 ADDRESSED STATES

The addressed state of a device determines how that device participates in a data transfer. A device can be:

- Deaddressed
- Addressed as listener
- Addressed as talker

When a device is deaddressed it does not participate in data transfers. A device addressed as listener can receive data, and a device addressed as talker can transmit data. On the IEC/IEEE bus, all devices addressed as listener can receive data simultaneously. However, only one device at a time can transmit data. Only after a data transfer has completed can another device be addressed as talker and transmit data. Thus, at a given time, there can be several devices addressed as listener but only one addressed as talker.

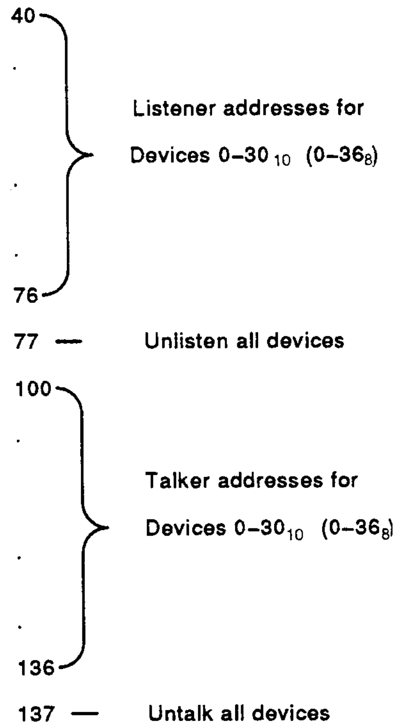
If an IEX11 is the controller-in-charge, it can change its own addressed state through the use of auxiliary commands listen only (**lon**) and talk only (**ton**). If the IEX11 is not controller-in-charge, its addressed state can be changed by being addressed as a talker or a listener. Only the controller-in-charge can change the addressed state of other devices on the bus.

The handshake logic of the IEC/IEEE bus ensures that each device addressed as listener has received the current data byte before the next byte is transferred. Thus if a device is addressed as listener, it must participate in the data transfer or else the transfer will hang. This means that if an IEX11 unit is addressed as listener, and if a read request has not been issued, then only the first byte will be transferred. When the read request is issued, the rest of the data transfer will occur.

1.4.1 IEC/IEEE Bus Device Addresses

Each device on the IEC/IEEE bus has a unique device address (0 - 36_8), through which it is referenced. The device address is specified in the low order five bits on an IEC/IEEE bus address command. Bits five and six are used to represent the type of command (Listener Address or Talker Address Group). The last address of each group is reserved to deaddress devices. This results in the following address representation (Listener and Talker address and device numbers are in octal):

Figure 1–1 IEC/IEEE Bus Device Addresses



The IO$_COMMANDS function is used to issue address commands. A typical sequence of address commands is:

Example 1–1 Address Commands Sequence

```
77   Unlisten all devices   (1)
137  Untalk                 (2)
41   Device 1 to listen     (3)
105  Device 5 to talker     (4)
```

1.4.2 Secondary Addresses

In addition to its primary address, each device may also have a number of secondary addresses (extended addressing). Normally, secondary addressing is used to select different functions within a device. Secondary addresses are specified by first sending a device its primary address and then one of its secondary addresses. Secondary addresses must be in the range 140_8 - 176_8. Refer to Appendix C.

The IEX11 hardware issues a DAC holdoff upon receipt of a secondary address. The application must then issue an auxiliary **dacr** command, release DAC holdoff, to release the bus so data can then be transferred. Refer to Section 3.3.1 and Section 5.2.2 for information on the issuing of auxiliary commands.

Note: The IEX11 has the ability to recognize both primary and secondary addressing modes. An application should use only one mode, primary or secondary, when addressing any IEX11 device.

1.5 DATA TRANSFERS

There are three ways in which a listener can detect the completion of a data transfer:

1. Byte count overflow
2. Match character termination
3. EOI termination

Byte count overflow occurs when the specified number of data bytes has been received. This method should only be used when the receiver knows the exact number of bytes that will be transferred.

A match character termination occurs when the specified match character has been consecutively received a specified number of times. This method is normally used with measuring devices which transfer data in ASCII records. For such applications, the match character could be a carriage return or a line feed.

The most versatile method of data transfer is accomplished through the use of EOI termination. With this method the EOI line is asserted during the transfer of the last byte. This method allows the transfer of variable length records with no restrictions on the data contents.

Note: The EOI termination method of data transfer involves additional software overhead during the transfer of the last data byte, which increases the time required for the transfer.

There are two methods by which the controller-in-charge can determine that a data transfer between two other devices has completed:

1. It can participate in the data transfer,
2. One, or more, of the devices which participate in the data transfer, can signal that the data transfer is complete by issuing a service request.

1.6 SERVICE REQUEST AND SERIAL POLL

One of the powerful features of the IEC/IEEE bus is the ability of the devices on the bus to asynchronously notify the controller-in-charge that an important event has been detected or that a device requires processing of some sort. This feature is implemented through the service request and serial poll. Typical situations in which a service request is issued would be if the tolerance values for a controller have been exceeded, or if the controller wishes to indicate that a data buffer is available for transfer.

In order to request service, a device loads a status byte in which bit six is set into its serial poll register. Bit six of the serial poll register of all devices is ORed with the SRQ line. If this bit in any device is set, then the service request (SRQ) line is asserted.

Note: An IEX11 unit which is controller-in-charge can be programmed to generate an interrupt when the SRQ becomes asserted.

Normally, when an application program detects that a service request is outstanding, it commands the controller-in-charge to issue a serial poll to determine which devices require servicing. During a serial poll, selected devices are addressed as talker and their serial poll register is read by the controller-in-charge. By examining bit six of the status byte of each device polled, the controller-in-charge can determine whether that device issued a service request. The other seven bits of this byte are available for coding information related to the service required.

A device on the IEC/IEEE bus sends its serial poll status byte when it is addressed as talker in the serial poll sequence. If the device issued a service request, then it will normally reload the serial poll register with bit six reset. If that was the only device requesting service, then the SRQ is also reset. If another device was simultaneously requesting service, then the service request line remains set, and the controller-in-charge continues with the serial poll. The only means of resetting the SRQ line is through a serial poll.

Note: An IEX11 unit which is controller-in-charge can be programmed to perform a serial poll when a service request interrupt occurs.

1.6.1 Parallel Poll

An alternative and faster polling method is the Parallel Poll. In a parallel poll, the controller-in-charge can obtain device status from up to eight devices through one bus request. Before a parallel poll can be performed, the selected devices must be configured to respond to a parallel poll. In this sequence the controller-in-charge addresses the selected device and sends it a configure byte, which indicates how the device should respond to a parallel poll.

The configure byte has the following format:

Bits	Meaning
0 - 2	Specifies the bit (0 - 7) that the device will respond to during a parallel poll.
3	Specifies the state (0 or 1) of the bit in the parallel poll register if the state of the device is true.

The controller-in-charge may command a device to cease responding to a parallel poll by sending it a parallel poll disable command. At any given time, only eight out of the possible maximum of 31 devices may be configured to respond to a parallel poll. This subset may be varied by using the unconfigure command to remove devices from the subset, and then the configure command to include other devices.

1.7 PROGRAMMING THE IEC/IEEE DEVICES

Before you can write a program to exchange data between an IEX11 interface and another device on the IEC/IEEE bus, you must know the procedure by which the device transfers data. First, you must know the IEC/IEEE bus address of the device. On many devices the IEC/IEEE address can be manually selected through switches on the device. Other devices have a fixed address that must be used. The IEX11 bus addresses are set through the IO$_INITIALIZE request. Each device on the bus must have a unique address.

You must determine if the IEX11 is to be the controller-in-charge. This is the basic factor determining how the IEX11 is to be programmed. in the majority of applications, the IEX11 will be used as controller-in-charge. The controller-in-charge is invariably a computer, and the IEX11 must be controller-in-charge for all applications where the only other devices on the IEC/IEEE bus are instruments that perform only basic input/output. The IEC/IEEE bus could also be used to exchange data with another computer. In this application, the IEX11 may or may not be controller-in-charge.

The next step in programming an application is to establish the protocol by which data will be exchanged. You should read the literature describing the device to be programmed. When the other device is a measuring instrument, the first step often involves setting the device to the remote state. The next step is often that of sending measurement parameters to the instrument. This is normally done by addressing the device as listener and sending it a series of ASCII strings using the IO$_WRITEPBLK request with the IO$M_EOI modifier. To read data from a device, it is often sufficient to address the device as talker and then read a buffer of data using the IO$_READPBLK request. Typically, you will have to program the service request/serial poll facility, but the use of parallel poll is not common in IEC/IEEE applications. You can expect that the manner in which the IEX11 is programmed will be dictated by the other devices on the bus. Thus, a good approach is to determine the procedure by which these devices exchange data and to program the IEX11 accordingly. The IEX11 and the driver support all of the IEC/IEEE features, and there should be no problem in programming them for every possible IEC/IEEE application.

The IEX Exerciser Program may be useful in helping you to establish the procedure necessary for the application. Normally, it will relieve you of the necessity of having to write and debug a test program. By using the IEX Exerciser Program, you can quickly issue a series of commands to the device and see how it responds. The IEX Exerciser Program can be used in a dialogue mode, and can also accept commands from an indirect command file. Refer to Chapter 5 and Appendix A for usage of the IEX Exerciser Program. Once the series of commands necessary to service the device has been determined, they can be incorporated into an application program. The commands in the IEX Exerciser Program correspond directly into requests to the driver.

1.7.1 Typical Application

A typical application of the IEX11 in a laboratory environment involves sending parameters to a device and then receiving measurement data from that device. Programming this application could be accomplished by utilizing the following requests to the IEX-VMS-Driver.

1. The IO$_INITIALIZE request is issued to set the IEX11 unit to system controller and to the CACS state. The IEC/IEEE bus address of the IEX11 controller is also established by this request.
2. The device to be a listener is addressed through the use of the IO$_COMMAND request.
3. The IO$_SETEVENT request is issued to instruct the driver that the program desires notification of service requests.
4. An IO$_WRITEPBLK!IO$M_EOI request is issued to send the parameter data to the listening device.
5. The IO$_REC_EVENT request is issued to wait for a service request.
6. When a service request is generated by the device, the program issues an IO$_SER_POLL request to perform a serial poll.
7. The device is unlistened through the use of the IO$_COMMAND request.
8. The device is addressed as talker through an IO$_COMMAND request.
9. An IO$_READPBLK request is issued to receive data from the talking device.
10. Before the process exits, the IO$_SETEVENT request (with a mask argument of zero) is issued to disable event recognition.

2 INSTALLATION

This chapter describes how to install the IEX-VMS-Driver from a tape cartridge, diskette, or CDROM. Instructions for running the Installation Verification Procedure (IVP) are also provided. A sample dialogue of the installation procedure is included at the end of this chapter. Refer to Appendix D for a description of the contents of the kit.

2.1 INSTALLATION PREREQUISITES

Before installing the IEX-VMS-Driver, check the contents of your kit against the bill of materials included in your shipment. If any items are missing, do not proceed with the installation.

Check that you have all of the following prerequisites before beginning the installation procedure.

- A supported VAX, MicroVAX, or VAXstation computer system. Refer to the IEX-VMS-Driver SPD/SSA for a list of supported computer systems
- The MicroVMS operating system, Version 4.7, or the VMS Version 5.0 or later operating system
- An IEX11 installed in the card cage
- 1300 free disk blocks during installation, 1000 permanent blocks
- The IEX-VMS-Driver on tape cartridge, diskette, or CDROM

Before installing the driver, the IEX11 module should be installed in the machine. You must know the address of the IEX11 as represented on its address switches. The installation procedure prompts you for this address.

This software is not supported on all processors. If you attempt to install this software on an unsupported processor, an error message will be displayed and the installation will be terminated. If this occurs, please refer to the IEX-VMS-Driver System Support Addendum (SSA) for the list of supported processors.

On VMS V5.0 or later systems, you must install the license registration Product Authorization Key (PAK) before installing the software. From the SYSTEM account, type the following at the "$" prompt:

```
$  @SYS$UPDATE:VMSLICENSE
```

This command file prompts you for all the information needed to install the PAK. Refer to the paper PAK sheets shipped with the software for all the necessary information. Full information on the license registration procedure is found in the *VMS License Management Utility Reference Manual*.

2.2 INSTALLATION TIME

The entire installation procedure takes about 15 minutes.

2.3 CSR AND VECTOR DETERMINATION

Prior to physically installing the IEX11 module, the base address and vector address switches must be set. The default address of the first IEX11 is 764100. The addresses of all additional IEX11's can be set anywhere in the floating space as long as the selected addresses do not conflict with the addresses of other devices or addresses reserved by Autoconfigure. The remainder of this section deals with vector address determination.

2.3.1 Device Mapping Known

Typically, the system manager or Customer Service representative will know where the next device to be installed in the system should be addressed and what the vector should be. Setting the base address and vector switches to those values will ensure smooth and effortless installation and testing of the IEX11 product.

2.3.2 Device Mapping Unknown

If the configuration of the target system is unknown, there are two methods for determining the correct vector assignments.

2.3.2.1 Determining Vectors with SYSGEN

Using the SYSGEN utility, it is possible to determine the address and vector assignments by using the CONFIG command to enter all of the devices currently installed as well as the not-yet installed IEX11. Refer to the *VAX/VMS System Generation Utility Reference Manual* for instructions on how to do this. Using SYSGEN to determine a configuration in no way affects the existing system configuration. To implement the configuration, the vector switches on the IEX11 must be set to the vector determined by SYSGEN.

Note: Adding an IEX11 to an existing system may cause SYSGEN to reassign vectors for other modules in the system.

2.3.2.2 Determining Vectors Using Autoconfigure

If you are installing on a system with an unknown configuration which uses Autoconfigure on system startup, or if you do not mind deinstalling the IEX11 module, there is an alternative method that can be taken to install this product:

1. Install the IEX11 module with a CSR setting of 764100, ignoring the vector setting (SET VECTOR > 400)
2. Install the IEX-VMS-Driver normally, using the correct device address of 764100 and the default value for the vector (300).
3. Following installation, reboot the system.
4. Using SYSGEN, issue a "SHOW/CONFIG" command and note the vector values for the devices in the system.
5. Shut down the system
6. Power down the system, disconnect the power cord, and remove the IEX11

Note: NEVER REMOVE a module from the system with the power on.

7. Set the vector on the IEX11 to the value specified by SYSGEN and reinstall the module. If the vector assignments for any other modules in the system were changed as a result of adding the IEX11, set the vector assignments for those devices at this time.
8. Reboot the system
9. Edit IEX$LOAD.COM in directory SYS$MANAGER on VMS V4.7 or in directory SYS$STARTUP on VMS V5.0 or later systems to reflect the correct vector settings for future reference.

2.4 INSTALLATION PROCEDURE

The IEX-VMS-Driver is installed using the VMSINSTAL command. This command can only be issued from the system manager's account.

If you are running MicroVMS Version 4.7, logging into a system manager's account may invoke the Manager menu. This installation procedure instructs you to first exit the menu. If your system account LOGIN.COM file does not invoke the Manager menu, or if you are running VMS V5.0 or later, the Manager menu is not invoked and you should proceed from step 3 below.

To install the IEX-VMS-Driver on a system running MicroVMS Version 4.7, using option 1 on the Manager menu, take the following steps:

Step 1. Invoke VMSINSTAL.

1 Log in to a SYSTEM account.

The system displays a welcome message and the Manager Menu:

```
       Welcome to MicroVMS V4.7

   Last interactive login on Xday, dd-mmm-yyyy hh:mm

Main Menu

        1 - Exit to DCL
        2 - Log out of the SYSTEM account
        3 - Invoke the MAIL utility
        4 - Invoke the PHONE utility
        5 - Add a user account to the system
        6 - Install optional software
        .
        .
        .
       14 - SHUT DOWN the system

Enter a number (? or ?# for HELP):  1

If you wish to use the menus again, type the following:
        $ @SYS$MANAGER:MGRMENU
```

2 Select option 1. The system displays the DCL prompt.

3 Set your default directory to SYS$UPDATE:

```
$  SET DEFAULT SYS$UPDATE
```

4 Type the following command line to run the VMSINSTAL utility:

```
$  @VMSINSTAL iex042 device_name OPTIONS N
```

If you are installing the device driver from a tape cartridge, specify the device_name in the preceding command line as the physical tape drive unit. If you are installing the device driver from a diskette or CDROM, specify the device_name in the preceding command line as the physical disk drive.

Specifying the OPTIONS N parameter causes VMSINSTAL to prompt you about the *Release Notes*. Print the *Release Notes* and read them before proceeding further with the installation. The release notes may contain last-minute instructions or changes to the installation procedures described in this guide.

The system displays the following informational messages:

```
  VAX/VMS Software Product Installation Procedure 4.7

It is dd-mmm-yyyy at hh:mm.
Enter a question mark (?) at any time for help.
```

The VMSINSTAL utility proceeds by displaying the following prompts:

```
* Are you satisfied with the backup of your system disk [YES]?
```

If you are satisfied with the backup of your system disk, press RETURN after this prompt. If you do not have a current backup of your system disk, answer NO to terminate this installation procedure. Back up your system disk and then begin the installation again. If your SYSTEM disk fails for any reason, you will need a copy of your previous system disk.

For information about the backup procedure, see the *MicroVMS V4.7 Release Notes*, and *Installing MicroVMS on MicroVAX II From a Tape Cartridge*, *Installing MicroVMS on MicroVAX I/II From Diskettes*.

Step 2. Load the distribution medium into the drive.

If you are installing the IEX-VMS-Driver from a tape cartridge, VMSINSTAL displays the following instruction and prompt:

```
Please mount the first volume of the set on MUA0:.
* Are you ready?
```

If you are installing the IEX-VMS-Driver from a diskette, VMSINSTAL displays the following instruction and prompt:

```
Please mount the first volume of the set on DUB1:.
* Are you ready?
```

Load the distribution media in the appropriate place.

Take the following steps to load or unload a tape cartridge:

1. If the cartridge-release handle of the tape drive is not already in the down or right position (depending on whether the drive is in the horizontal or vertical position), put it in this position.
2. Check that the red light (the LOAD/UNLOAD switch) on the right of (or above) the tape drive is in the out position and not glowing.
3. Check that the small green (activity) light on the left of (or below) the tape drive is glowing steadily.
4. Lift the cartridge-release handle.

5 Insert the tape cartridge, with the arrow on the cartridge facing toward the drive (away from the Digital logo).

6 Put the cartridge-release handle down (or to the right) on the drive, and make sure that the handle is locked solidly in place.

7 Push the red light button in.

8 Check that you have successfully loaded the tape cartridge by seeing that the red light button is in and glowing steadily, and the green light is also glowing steadily.

Take the following steps to load a diskette:

- Open the diskette door by pressing on its outer edge.
- Align the orange arrow on the diskette with the orange stripe on the drive and insert the diskette. The write-protect notch is down in drive 1 and up in drive 2.
- After inserting the diskette, press the drive door to close it.

Enter YES after the ‘‘Are you ready?’’ prompt when you have loaded your distribution medium.

When your distribution medium has been successfully loaded into the drive, VMSINSTAL displays an informational message telling you that the medium has been mounted and that installation has begun.

```
%MOUNT-I-MOUNTED, IEX mounted on _MUA0:
The following products will be processed:
  IEX V4.2

        Beginning installation of IEX V4.2 at 11:38
%VMSINSTAL-I-RESTORE, Restoring product saveset A...
```

Step 3. Select the release notes option.

VMSINSTAL now prompts you to select a release notes option:

```
Release Notes Options:

        1. Display release notes
        2. Print release notes
        3. Both 1 and 2

* Select option [3]:
```

You can display the release notes on the screen, print them, or do both. Then, you can continue the installation procedure by entering YES after the following prompt:

```
Do you want to continue the installation [NO]? yes
```

The installation then proceeds with the following message.

```
%VMSINSTAL-I-RELMOVED, The products release notes have been successfully moved to SYS$HELP.
%IEX-I-VMSOK, Installing IXDRIVER V4.2 on VMS Version V4.7
```

VMSINSTAL prompts you to select if files replaced by this installation are to be purged. Answer YES to this question.

```
* Do you want to purge files replaced by this installation [YES]?:
```

On VMS V5.0 or later systems the VMSINSTAL procedure then displays product registration information and asks if you have registered and loaded the Product Authorization Key (PAK) which you received with your kit.

```
        Product:      IEX-VMS-Driver
        Producer:     DEC
        Version:      4.2
        Release Date: 01-JUN-1990

* Does this product have an authorization key registered and loaded? YES
```

If you have registered and loaded the Product Authorization Key for the IEX-VMS-Driver answer YES. The procedure will then continue.

If you have not registered and loaded the Product Authorization Key answer NO. VMSINSTAL will then print out the following informational messages and you will not be allowed to run the IVP.

```
  An authorization key for IEX-VMS has not been registered and loaded.

      The IEX-VMS IVP will NOT be run as part of this installation.
      After the IEX-VMS license has been registered and loaded, it
      is recommended that the IVP be run with the following command:

        $ RUN SYS$COMMON:[SYSTEST.IEX]IEX$IVP
```

Step 4. Run the Installation Verification Procedure

This question determines whether the Installation Verification Procedure (IVP) is executed. This question will not be asked on VMS V5.0 or later systems if you have not registered and loaded the Product Registration Key (PAK).

```
* Do you want to run the IVP after the installation [YES]? yes
```

You should run the IVP to verify that the installation of the device driver was successful. Do not run the IVP if you are unsure of the base address and vector address of the IEX11 installed in your system. Failure of the IVP when you do not know the address of your IEX11 does not indicate that the driver installation failed.

Step 5. Specify IEX11 Base Address and Vector Address

VMSINSTAL now prints the following informational message.

```
    The installation will check for the VMS driver already installed
    in the system.  If there is no driver installed in the system you
    will receive the following message:

    "%SYSTEM-W-NOSUCHDEV, no such device available"

    The message can be ignored, the installation will proceed normally.
    Do not run the IVP if you must edit IEX$LOAD.COM to insert the
    correct CSR and vector for your IEX11.

%SYSTEM-W-NOSUCHDEV, no such device available
```

In this example, there is no preexisting version of the IEX11 Device Driver, and the ''no such device available'' message is printed.

VMSINSTAL now prompts you for the base address and the vector address of each IEX11 that you have in your system. The IEX11 is a floating device and may or may not be configured at the factory configurations.

Each IEX11 physical interface board contains two IEC/IEEE bus controllers. Thus, if the default value of one interface is specified, the devices IXA0: and IXA1: are generated. If two interfaces are specified, IXB0: and IXB1: will be generated in addition to IXA0: and IXA1:.

The maximum number of IEX11 interfaces which can be entered is four, for a total of eight devices.

If you do not currently know the address setting(s) of the IEX11(s) in your system, use the default address settings in the installation prompt. This will accomplish the task of loading the driver, but to use the driver, the correct addresses must be be entered into the IEX$LOAD.COM file.

The VMSINSTAL procedure then asks how many IEX11's you have installed in your system.

```
* Number of IEX11 interfaces to be supported [1]: 1
```

On UNIBUS processors the VMSINSTAL procedure then prompts you for the UNIBUS adapter number for the interface, displaying the default UNIBUS adapter number for your system. Override the adapter number for each IEX11 if the physical interface is not connected to the default adapter. Refer to SYSGEN for further information on determining the adapter number.

```
* Enter UNIBUS adapter number for interface 1 [3]: 3
```

The VMSINSTAL procedure then prompts you for the addresses of the IEX11 devices. Enter the base address and vector address of the first IEX11 in your system.

Note: The user must specify the base address of the interface instead of the address of the control and status register (CSR), which is at offset 10_8 from the base. In the standard installation the CSR address is 764110, however, the base address, 764100, must be entered as the answer to the CSR address question.

The base address and vector address are ONLY used to generate the IEX$LOAD.COM file for use in loading more than two GPIB controllers at system startup or to bypass Autoconfigure under special conditions.

```
* Enter CSR address for interface 1 [764100]: 764100
* Enter vector address for interface [300]: 320
```

If you specified more than 1 IEX11, the VMSINSTAL program then prompts you for the base address and vector address of the second IEX11. This procedure continues until all IEX11's have been accounted for.

At this point, no further action is required of you. Informational messages are printed as the installation proceeds. Section 2.7 and Section 2.8 provide complete examples of installation procedures on MicroVMS V4.7 and VMS V5.0 systems with the informational messages printed.

2.5 INSTALLATION VERIFICATION PROCEDURE - IVP

The installation verification procedure checks to see that the IEX-VMS-Driver was successfully installed. A successful execution of the IVP is shown below. A loopback cable is required for successful execution of the IVP. The IVP only tests the first IEX11 (IXA0:, IXA1).

```
Beginning the IEX-VMS-Driver V4.2 Installation Verification Procedure
Copyright (C) Digital Equipment Corporation. 1990. All Rights Reserved.

Verification of IEX-VMS-Driver V4.2 was successful

Completion of the IEX-VMS-Driver V4.2 Installation Verification Procedure
```

The Installation Verification Procedure will test that the driver is properly installed and that there is an IEX11 at the address specified. The IVP can not test that the interrupt vector is correct.

If the IVP fails, the successful message is not printed and a failure message and error report is generated. The following guidelines can help to isolate the problem.

- If the IVP does not report information as shown above, and reports a ‘‘no such device’’ (NOSUCHDEV) error message, the base address and/or vector address specified for the IEX11 are probably incorrect. The driver was probably installed correctly, but since it cannot find a device at the address you specify, the IVP is not successful. Try rerunning the IVP after editing the IEX$LOAD.COM file to specify the correct base address.
- If you are certain that the base address was specified correctly, but the IVP still fails to report information above, the IEX11 may have failed or not be properly seated in the card cage. Run the IEX11 diagnostics to test that the IEX11 is properly seated in the card cage and is functioning.
- If the IVP does report information as shown above, but still reports an error message, the driver installation is unsuccessful. Try reinstalling the driver by starting over at the beginning.

2.6 POST-INSTALLATION TASKS

The IEX Exerciser Program (IEX.EXE) can be used to determine that the address and vector settings are correct. After installing the loopback cable on the IEX11, issue the following commands:

```
$  SET DEF SYS$COMMON:[SYSHLP.EXAMPLES.IEX]
$  RUN IEX
Copyright (c) Digital Equipment Corporation. 1989, 1990. All rights reserved.
$IEX> @IEX$DEMO
$IEX> EXIT
```

One of the first tests IEX$DEMO performs is to have one controller address the other controller as listener and wait for that command to complete. This test will generate an interrupt. If a timeout error is displayed on the terminal, the interrupt was not serviced and is an indication that the vector setting of the IEX11 is incorrect. If Autoconfigure is enabled, use SYSGEN to determine at what vector the IEX driver was loaded and verify that it matches the actual hardware setting. If it does not match, correct the hardware vector setting, following the directions in the *IEU11/IEQ11 Hardware User's Guide*.

The indirect command file IEX$DEMO.COM can be examined to help in generating a command file for providing a customized IEX11 confidence test.

Refer to Appendix A for a sample dialogue of the execution of IEX$DEMO.COM.

2.7 SAMPLE INSTALLATION DIALOGUE - MicroVMS V4.7

The following is a sample installation dialogue for installation of the IEX-VMS-Driver on a MicroVAX II system running MicroVMS 4.7.

```
Username: SYSTEM
Password:


    Last interactive login on Friday, 22-JAN-1990 11:36
    Last non-interactive login on Friday, 22-JAN-1990 11:37

$ show default
  SYS$SYSROOT:[SYSMGR]
$ set default sys$update
$ @vmsinstal IEX042 mua0:

        VAX/VMS Software Product Installation Procedure V4.7


It is 22-JAN-1990 at 11:38.
Enter a question mark (?) at any time for help.
* Are you satisfied with the backup of your system disk [YES]? yes

Please mount the first volume of the set on MUA0:.
* Are you ready? yes
%MOUNT-I-MOUNTED, IEX mounted on _MUA0:

The following products will be processed:

  IEX V4.2

        Beginning installation of IEX V4.2 at 11:38

%VMSINSTAL-I-RESTORE, Restoring product saveset A...
%VMSINSTAL-I-RELMOVED, The products release notes have been successfully moved to SYS$HELP.

$! Copyright (c) 1990 Digital Equipment Corporation.  All rights reserved.

%IEX-I-VMSOK, Installing IXDRIVER V4.2 on VMS Version V4.7
* Do you want to purge files replaced by this installation [YES]? yes

        The installation will check for the VMS driver already installed
        in the system.  If there is no driver installed in the system you
        will receive the following message:

        "%SYSTEM-W-NOSUCHDEV, no such device available"

        This message can be ignored, the installation will proceed normally.
        Do not run the IVP if you must edit IEX$LOAD.COM to insert the
        correct CSR and vector for your IEX11.

%SYSTEM-W-NOSUCHDEV, no such device available
* Do you want to run the IVP after the installation [YES]? yes

        The term IEX11 interface in this procedure stands either
        for the IEU11 interface for VAX UNIBUS systems or for
        IEQ11 for MicroVAX Q22 bus systems.

        When answering the CSR address question, the base address
        of the interface should be specified and not the address of the
        Control/status register which is at an offset of 10(8) from
        the base address.

        The first IEX should be at address 764100.  The vector will
        be assigned by Autoconfig to be in floating space (starts
        at 300).  All additional IEX modules will be given an
        address in floating address space.


* Number of IEX11 interfaces to be supported [1]: 1
* Enter CSR address for interface 1 [764100]: 764100
* Enter vector address for interface 1 [300]: 320

%VMSINSTAL-I-SYSDIR, This product creates system directory [SYSHLP.EXAMPLES.IEX].
```

```
       The driver will now be linked.  The linker will produce a
       warning message that the driver has no transfer address.
       Drivers do not have a transfer address,  and this message
       should be ignored.

%LINK-W-USRTFR, image VMI$ROOT:[SYSUPD.IEX042]IXDRIVER.EXE;1 has no user transfer address

       The command file IEX$LOAD.COM will be located in the
       directory SYS$SPECIFIC:[SYSMGR].  This command file can be
       used to load the IEX-VMS-Driver if the system must be
       rebooted.

       All sources and all demonstration programs will be stored
       in the directory SYS$COMMON:[SYSHLP.EXAMPLES.IEX]. This
       directory also contains the command file IEX$DEMONS.COM
       to build the demonstration programs.

       The Simplified User Interface (IEX$SUI) object will
       be stored in the directory referenced by SYS$LIBRARY.
       The source program will be stored in the directory
       SYS$COMMON:[SYSHLP.EXAMPLES.IEX].

       If the two controllers on the IEX11 are connected by a
       cable, the device exerciser program IEX can be run with
       the demonstration command file IEX$DEMO.COM by using the
       following command:

                 $ SET DEF SYS$COMMON:[SYSHLP.EXAMPLES.IEX]
                 $ RUN IEX
                 IEX$>@IEX$DEMO


  Note: The names and locations of the IEX-VMS files were changed
        in IEX-VMS V4.0.  If you are upgrading from a version prior
        to IEX-VMS V4.0 to IEX-VMS V4.0 or later and the old files
        are found, they will be renamed as follows:

   Prior to IEX-VMS V4.0     to  IEX-VMS V4.0 and later
   ========================      ================================
   SYS$MANAGER:IEXLOAD.COM       SYS$SPECIFIC:[SYSMGR]IEX$LOAD.COM
   SYS$SYSTEM:IXDRIVER.EXE       SYS$SPECIFIC:[SYSEXE]IXDRIVER.EXE
   SYS$MANAGER:IXDRIVER.COM      SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IXDRIVER.COM
   SYS$EXAMPLES:IXDRIVER.OPT     SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IXDRIVER.OPT
   SYS$EXAMPLES:IXDRIVER.MAR     SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IXDRIVER.MAR
   SYS$EXAMPLES:IEX.COM          SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX.COM
   SYS$EXAMPLES:IEX.EXE          SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX.EXE
   SYS$EXAMPLES:IEX.FOR          SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX.FOR
   SYS$EXAMPLES:IEXDEX.FOR       SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEX.FOR
   SYS$LIBRARY:IEXSUI.OBJ        SYS$LIBRARY:IEX$SUI.OBJ
   SYS$EXAMPLES:IEXSUI.MAR       SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$SUI.MAR
   SYS$EXAMPLES:IEXSUI.COM       SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$SUI.COM
   SYS$EXAMPLES:IEXFUNC.FOR      SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$FUNC.FOR
   SYS$EXAMPLES:IEXFUNC.MAR      SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$FUNC.MAR
   SYS$EXAMPLES:IEXFUNC.MLB      SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$FUNC.MLB
   SYS$EXAMPLES:IEXFUNC.PAS      SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$FUNC.PAS
   SYS$EXAMPLES:IEXFUNC.BAS      SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$FUNC.BAS
   SYS$EXAMPLES:IEXDEMO.COM      SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMO.COM
   SYS$EXAMPLES:IEXDEMOAF.FOR    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOAF.FOR
   SYS$EXAMPLES:IEXDEMOAF.EXE    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOAF.EXE
   SYS$EXAMPLES:IEXDEMOB.FOR     SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOB.FOR
   SYS$EXAMPLES:IEXDEMOB.EXE     SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOB.EXE
   SYS$EXAMPLES:IEXDEMOAM.MAR    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOAM.MAR
   SYS$EXAMPLES:IEXDEMOAM.EXE    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOAM.EXE
   SYS$EXAMPLES:IEXDEMOAS.FOR    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOAS.FOR
   SYS$EXAMPLES:IEXDEMOAS.EXE    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOAS.EXE
   SYS$EXAMPLES:IEXDEMOAB.BAS    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOAB.BAS
   SYS$EXAMPLES:IEXDEMOAB.EXE    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOAB.EXE
   SYS$EXAMPLES:IEXDEMOAP.PAS    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOAP.PAS
   SYS$EXAMPLES:IEXDEMOAP.EXE    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOAP.EXE
   SYS$EXAMPLES:IEXDEMONS.COM    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMONS.COM
   SYS$EXAMPLES:UETIX1100.MAR    SYS$COMMON:[SYSTEST.IEX]UETIX1100.MAR
   SYS$EXAMPLES:IEXIVP.MAR       SYS$COMMON:[SYSTEST.IEX]IEX$IVP.MAR
   SYS$EXAMPLES:IEXIVP.COM       SYS$COMMON:[SYSTEST.IEX]IEX$IVP.COM
```

```
   SYS$TEST:IEXIVP.EXE           SYS$COMMON:[SYSTEST.IEX]IEX$IVP.EXE

      Checking for files from IEX-VMS installations prior to IEX-VMS V4.0

%VMSINSTAL-I-MOVEFILES, Files will now be moved to their target directories...

        *------------------------------------------------*
        *    Installation Verification Procedure for    *
        *               IEX-VMS-Driver V4.2             *
        *------------------------------------------------*

        The Installation Verification Procedure will test that the
        driver is properly installed and that there is an IEX11 at
        the address specified.  It can not test that the interrupt
        vector is correct.

__Driver_____Start____End____Dev___DDB______CRB______IDB_____Unit__UCB____
IXDRIVER   802E2CA0 802E5740
                             IXA 804B7BC0 80444090 804B7B00
                                                                0 802D4340
                                                                1 802E5740

Device                  Device           Error
 Name                   Status           Count
IXA0:                   Online               0
IXA1:                   Online               0

Beginning the IEX-VMS-Driver V4.2 Installation Verification Procedure
Copyright (C) Digital Equipment Corporation. 1990. All Rights Reserved.

Verification of IEX-VMS-Driver V4.2 was successful

Completion of the IEX-VMS-Driver V4.2 Installation Verification Procedure


        Installation of IEX V4.2 completed at 11:43

        VMSINSTAL procedure done at 11:43

$ logo

  SYSTEM       logged out at 22-JAN-1990 11:44
```

2.8 SAMPLE INSTALLATION DIALOGUE - VMS V5.0

The following is a sample installation dialogue for installation of the IEX-VMS-Driver under VMS 5.0.

```
Username: SYSTEM
Password:


    Last interactive login on Friday, 22-JAN-1990 11:36
    Last non-interactive login on Friday, 22-JAN-1990 11:37

$ show default
  SYS$SYSROOT:[SYSMGR]
$ set default sys$update
$ @vmsinstal IEX042 mua0:

        VAX/VMS Software Product Installation Procedure V5.0


It is 22-JAN-1990 at 11:38.
Enter a question mark (?) at any time for help.
* Are you satisfied with the backup of your system disk [YES]? yes

Please mount the first volume of the set on MUA0:.
* Are you ready? yes
%MOUNT-I-MOUNTED, IEX mounted on _MUA0:

The following products will be processed:

  IEX V4.2

        Beginning installation of IEX V4.2 at 11:38

%VMSINSTAL-I-RESTORE, Restoring product saveset A...
%VMSINSTAL-I-RELMOVED, The products release notes have been successfully moved to SYS$HELP.

$! Copyright (c) 1990 Digital Equipment Corporation.  All rights reserved.

%IEX-I-VMSOK, Installing IXDRIVER V4.2 on VMS Version V5.0
* Do you want to purge files replaced by this installation [YES]? yes

        Product:      IEX-VMS-Driver
        Producer:     DEC
        Version:      4.2
        Release Date: 01-JUN-1990


* Does this product have an authorization key registered and loaded? YES

        The installation will check for the VMS driver already installed
        in the system.  If there is no driver installed in the system you
        will receive the following message:

        "%SYSTEM-W-NOSUCHDEV, no such device available"

        This message can be ignored, the installation will proceed normally.
        Do not run the IVP if you must edit IEX$LOAD.COM to insert the
        correct CSR and vector for your IEX11.

%SYSTEM-W-NOSUCHDEV, no such device available
* Do you want to run the IVP after the installation [YES]? yes

        The term IEX11 interface in this procedure stands either
        for the IEU11 interface for VAX UNIBUS systems or for
        IEQ11 for MicroVAX Q22 bus systems.

        When answering the CSR address question, the base address
        of the interface should be specified and not the address of the
        Control/status register which is at an offset of 10(8) from
        the base address.
```

```
        The first IEX should be at address 764100.  The vector will
        be assigned by Autoconfig to be in floating space (starts
        at 300).  All additional IEX modules will be given an
        address in floating address space.

* Number of IEX11 interfaces to be supported [1]: 1
* Enter CSR address for interface 1 [764100]: 764100
* Enter vector address for interface 1 [300]: 320

%VMSINSTAL-I-SYSDIR, This product creates system directory [SYSHLP.EXAMPLES.IEX].

        The driver will now be linked.  The linker will produce a
        warning message that the driver has no transfer address.
        Drivers do not have a transfer address,  and this message
        should be ignored.

%LINK-W-USRTFR, image VMI$ROOT:[SYSUPD.IEX042]IXDRIVER.EXE;1 has no user transfer address

        The command file IEX$LOAD.COM will be located in the
        directory SYS$SPECIFIC:[SYS$STARTUP]. This command file can be
        used to load the IEX-VMS-Driver if the system must be
        rebooted.

        All sources and all demonstration programs will be stored
        in the directory SYS$COMMON:[SYSHLP.EXAMPLES.IEX]. This
        directory also contains the command file IEX$DEMONS.COM
        to build the demonstration programs.

        The Simplified User Interface (IEX$SUI) object will
        be stored in the directory referenced by SYS$LIBRARY.
        The source program will be stored in the directory
        SYS$COMMON:[SYSHLP.EXAMPLES.IEX].

        If the two controllers on the IEX11 are connected by a
        cable, the device exerciser program IEX can be run with
        the demonstration command file IEX$DEMO.COM by using the
        following command:

                  $ SET DEF SYS$COMMON:[SYSHLP.EXAMPLES.IEX]
                  $ RUN IEX
                  IEX$>@IEX$DEMO


  Note: The names and locations of the IEX-VMS files were changed
        in IEX-VMS V4.0.  If you are upgrading from a version prior
        to IEX-VMS V4.0 to IEX-VMS V4.0 or later and the old files
        are found, they will be renamed as follows:

   Prior to IEX-VMS V4.0     to  IEX-VMS V4.0 and later
   ========================      ================================
   SYS$MANAGER:IEXLOAD.COM       SYS$SPECIFIC:[SYS$STARTUP]IEX$LOAD.COM
   SYS$SYSTEM:IXDRIVER.EXE       SYS$SPECIFIC:[SYS$LDR]IXDRIVER.EXE
   SYS$MANAGER:IXDRIVER.COM      SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IXDRIVER.COM
   SYS$EXAMPLES:IXDRIVER.OPT     SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IXDRIVER.OPT
   SYS$EXAMPLES:IXDRIVER.MAR     SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IXDRIVER.MAR
   SYS$EXAMPLES:IEX.COM          SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX.COM
   SYS$EXAMPLES:IEX.EXE          SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX.EXE
   SYS$EXAMPLES:IEX.FOR          SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX.FOR
   SYS$EXAMPLES:IEXDEX.FOR       SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEX.FOR
   SYS$LIBRARY:IEXSUI.OBJ        SYS$LIBRARY:IEX$SUI.OBJ
   SYS$EXAMPLES:IEXSUI.MAR       SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$SUI.MAR
   SYS$EXAMPLES:IEXSUI.COM       SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$SUI.COM
   SYS$EXAMPLES:IEXFUNC.FOR      SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$FUNC.FOR
   SYS$EXAMPLES:IEXFUNC.MAR      SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$FUNC.MAR
   SYS$EXAMPLES:IEXFUNC.MLB      SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$FUNC.MLB
   SYS$EXAMPLES:IEXFUNC.PAS      SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$FUNC.PAS
   SYS$EXAMPLES:IEXFUNC.BAS      SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$FUNC.BAS
   SYS$EXAMPLES:IEXDEMO.COM      SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMO.COM
   SYS$EXAMPLES:IEXDEMOAF.FOR    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOAF.FOR
   SYS$EXAMPLES:IEXDEMOAF.EXE    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOAF.EXE
   SYS$EXAMPLES:IEXDEMOB.FOR     SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOB.FOR
   SYS$EXAMPLES:IEXDEMOB.EXE     SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOB.EXE
   SYS$EXAMPLES:IEXDEMOAM.MAR    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOAM.MAR
```

```
   SYS$EXAMPLES:IEXDEMOAM.EXE    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOAM.EXE
   SYS$EXAMPLES:IEXDEMOAS.FOR    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOAS.FOR
   SYS$EXAMPLES:IEXDEMOAS.EXE    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOAS.EXE
   SYS$EXAMPLES:IEXDEMOAB.BAS    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOAB.BAS
   SYS$EXAMPLES:IEXDEMOAB.EXE    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOAB.EXE
   SYS$EXAMPLES:IEXDEMOAP.PAS    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOAP.PAS
   SYS$EXAMPLES:IEXDEMOAP.EXE    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMOAP.EXE
   SYS$EXAMPLES:IEXDEMONS.COM    SYS$COMMON:[SYSHLP.EXAMPLES.IEX]IEX$DEMONS.COM
   SYS$EXAMPLES:UETIX1100.MAR    SYS$COMMON:[SYSTEST.IEX]UETIX1100.MAR
   SYS$EXAMPLES:IEXIVP.MAR       SYS$COMMON:[SYSTEST.IEX]IEX$IVP.MAR
   SYS$EXAMPLES:IEXIVP.COM       SYS$COMMON:[SYSTEST.IEX]IEX$IVP.COM
   SYS$TEST:IEXIVP.EXE           SYS$COMMON:[SYSTEST.IEX]IEX$IVP.EXE

      Checking for files from IEX-VMS installations prior to IEX-VMS V4.0

%VMSINSTAL-I-MOVEFILES, Files will now be moved to their target directories...

        *-----------------------------------------------*
        *    Installation Verification Procedure for    *
        *               IEX-VMS-Driver V4.2             *
        *-----------------------------------------------*


        The Installation Verification Procedure will test that the
        driver is properly installed and that there is an IEX11 at
        the address specified.  It can not test that the interrupt
        vector is correct.

__Driver_____Start____End____Dev___DDB______CRB______IDB_____Unit__UCB____
IXDRIVER   802E2CA0 802E5740
                             IXA 804B7BC0 80444090 804B7B00
                                                               0 802D4340
                                                               1 802E5740

Device                  Device           Error
 Name                   Status           Count
IXA0:                   Online               0
IXA1:                   Online               0


Beginning the IEX-VMS-Driver V4.2 Installation Verification Procedure
Copyright (C) Digital Equipment Corporation. 1990. All Rights Reserved.

Verification of IEX-VMS-Driver V4.2 was successful

Completion of the IEX-VMS-Driver V4.2 Installation Verification Procedure


        Installation of IEX V4.2 completed at 11:43

        VMSINSTAL procedure done at 11:43

$ logo

  SYSTEM       logged out at 22-JAN-1990 11:44
```

3 SOFTWARE INTERFACE - FUNCTION CODES AND I/O STATUS BLOCKS

The IEX-VMS-Driver is invoked using the SYS$QIO system service call. Complete details on the SYS$QIO system service routine can be found in the *VAX/VMS System Services Manual*.

Prior to IEX-VMS-Driver Version 3.0, access to the driver was restricted to processes which had physical I/O privileges. Beginning with Version 3.0, new nonprivileged function codes were created. Driver specific function codes which have a trailing **$** in their symbolic names do not require physical I/O privileges. Those which do not have a trailing **$** still require physical I/O privileges.

The driver is controlled by passing function codes and function code modifiers to the SYS$QIO service routine. The following sections describe the function codes and function modifiers for this service routine.

3.1 DEFINITION FILES

The definitions of the function codes and modifiers are contained in language specific files. Your program should include the definition file appropriate for your language. The definition file is named IEX$FUNC.*, where * is the extension appropriate for your language. The language extension names are shown in Table 3–1. The IEX-VMS-Driver also uses some function codes which are defined in the standard VMS system I/O definitions. To include all of these function codes into a FORTRAN program, add the following line of code:

```
INCLUDE 'IEX$FUNC.FOR'          !IEX function codes
INCLUDE '($IODEF)'              !Standard VMS function codes
```

Table 3–1 Definition Files

File Extension	Language
IEX$FUNC.BAS	BASIC
IEX$FUNC.FOR	Fortran
IEX$FUNC.MLB	Macro Library
IEX$FUNC.PAS	Pascal

Note: When you include the IEX$FUNC file, this file should appear first in any list of included files you have. They are located in SYS$COMMON:[SYSHLP.EXAMPLES.IEX].

3.2 I/O STATUS BLOCK FORMAT

I/O status blocks (IOSB) are returned for all SYS$QIO calls. Each IEX-VMS-Driver function code returns two longwords of information in the IOSB.

For many of the IEX I/O requests, the driver returns information on the state of the IEX11 controller at the end of the I/O request in addition to returning the status of the call. This information may be extremely useful in determining why a program is failing.

It is the responsibility of the application to check both the SYS$QIO system service status, as well as the return status field in the IOSB to ensure the integrity of the SYS$QIO call.

A typical format of the I/O Status Buffer for a SYS$QIO request is:

Figure 3–1 I/O Status Block Format

*				return status
reserved	addressed state	*	contr state	IEC/IEEE status register
31	27	23	19	15 0

* Refer to the specific function for the exact content of the IOSB.

The first word of the IOSB always returns the status of the call. Following are the return status messages specific to the IEX11 driver. Other error codes are returned from VMS and can be decoded using LIB$SIGNAL. These condition values and the corresponding symbolic definitions are included in the default system library (SYS$LIBRARY:STARLET.OLB). A listing of these symbolic codes can be obtained by invoking the system macro $SSDEF. In a FORTRAN program, these VMS symbolic codes can be included by adding the following command before the declarations in the program:

```
INCLUDE      '($SSDEF)'
```

It is recommended that these symbolic names be used rather than the corresponding numeric values when testing for specific error conditions within an application program.

The standard VMS I/O status codes returned in the low word of the first longword of the IOSB and in the SYS$QIO call are shown in Table 3–2.

Table 3–2 Status Codes Returned by Driver

Symbolic Name	Value (hex)	Symbolic Name	Value (hex)	Symbolic Name	Value (hex)
SS$_ACCVIO	0C	SS$_DEVICEFULL	850	SS$_MSGNOTFND	621
SS$_BADPARAM	14	SS$_DEVINACT	20D4	SS$_NOPRIV	24
SS$_CANCEL	830	SS$_DEVOFFLINE	84	SS$_NORMAL	01
SS$_DATACHECK	5C	SS$_DEVREQERR	334	SS$_NOSUCHDEV	908
SS$_DEVACTIVE	2C4	SS$_ILLIOFUNC	F4	SS$_POWERFAIL	364
SS$_DEVALRALLOC	641	SS$_IVADDR	134	SS$_TIMEOUT	22C

The high word of the first longword of the IOSB contains function specific information.

The low word of the second longword of the IOSB contains the value of the IEC/IEEE Status Register at the completion of the I/O request. This register gives information on the state of the IEX11 controller and the IEEE bus.

Table 3–3 contains a description of the bits of this register and their meaning when the bits are set.

Table 3–3 IEC/IEEE Status Register Format

Bit Position	Mnemonic	Description
0	ULPA	LSB of the last address recognized by the TMS 9914A
1	TADS (or TACS)	Device is addressed to talk
2	LADS (or LACS)	Device is addressed to listen
3	TPAS	TMS 9914A is in the talker primary addressed state
4	LPAS	TMS 9914A is in the listener primary addressed state
5	ATN	The attention line is low (TRUE) on the bus
6	LLO	Local lockout is in operation
7	REM	Device is in the remote state
8	REN	The remote enable line is low (TRUE)
9	IFC	The interface clear line is low (TRUE) *
10	SRQ	The service request line is low (TRUE)
11	EOI	The end or identify line is low (TRUE)
12	NRFD	The not ready for data line is low (TRUE)
13	NDAC	The not data accepted line is low (TRUE)
14	DAV	The data available line is low (TRUE)
15	ATN	The attention line is low (TRUE)

***The IFC bit of this register does not indicate a true value if the device is a system controller using the sic (send interface clear) auxiliary command.**

The high word of the second longword of the IOSB contains function specific information. Typically the first four bits of the low byte contain information on the controller state, with one of the following values being returned:

1 - Controller Idle State
2 - Controller Active State
3 - Controller Standby State

Typically the first four bits of the high byte contain information on the addressed state of the IEX11 unit, with one of the following values being returned:

0 - not addressed
1 - addressed as listener
2 - addressed as talker

Refer to the specific function to determine the exact format of the IOSB for that function.

3.3 FUNCTION CODES, ARGUMENTS, AND MODIFIERS

The following sections list the function codes, their arguments, and the function modifiers that can be used with each mode of operation. Table 3–4 shows a summary of all available functions.

Table 3–4 Function Codes Summary

Function Code	Modifier	Value (hex)	Parameters
IO$_AUXILIARY		0A	P1 = command
IO$_AUXILIARY$		2B	P1 = command
IO$_COMMAND		07	P1 = command
IO$_COMMAND$		28	P1 = command
IO$_COMMANDS		12	P1 = bufadr, P2=bytcnt
IO$_COMMANDS$		2C	P1 = bufadr, P2=bytcnt
IO$_GO_TO_CACS		08	None
IO$_GO_TO_CACS$		29	None
IO$_GO_TO_CSBS		09	None
IO$_GO_TO_CSBS$		2A	None
IO$_INITIALIZE*		04	P1 = mask, P2 = address
IO$_INITIALIZE$		24	P1 = mask, P2 = address
IO$_LOADPARPOLL		14	P1 = status byte
IO$_LOADPARPOLL$		33	P1 = status byte
IO$_PAR_POLL		06	None
IO$_PAR_POLL$		26	None
IO$_PARPOLLCON		13	P1 = bufadr, P2 = bytcnt
IO$_PARPOLLCON$		32	P1 = bufadr, P2 = bytcnt
IO$_PASSCONTROL		0F	P1 = talker
IO$_PASSCONTROL$		2F	P1 = talker
IO$_READ{L,P,V}BLK*		{21,0C,31}	P1 = bufadr, P2 = bytcnt, [P3 = match char], [P4 = match char cnt], [P5 = talker address]
	IO$M_TCS	40	
	IO$M_ENA_SRQ	100	
	IO$M_CONTROL	200	
	IO$M_STANDBY	400	
	IO$M_LEAVIS	800	
IO$_REC_EVENT		0E	None

*These are standard VMS function codes and modifiers.

Table 3–4 (Cont.) Function Codes Summary

Function Code	Modifier	Value (hex)	Parameters
	IO$M_NOWAIT	80	
IO$_REC_EVENT$		2E	None
	IO$M_NOWAIT	80	
IO$_SENSEMODE*		27	None
IO$_SER_POLL		02	P1 = tlkbuf, P2 = bytcnt, P3 = response buffer
	IO$M_ALLDEVICES	40	
IO$_SER_POLL$		22	P1 = tlkbuf, P2 = bytcnt, P3 = response buffer
	IO$M_ALLDEVICES	40	
IO$_SERVICE		01	P1 = status byte, [P2 = extended address]
IO$_SERVICE$		25	P1 = status byte, [P2 = extended address]
IO$_SETEVENT		0D	P1 = event mask
IO$_SETEVENT$		2D	P1 = event mask
IO$_SETMODE*		23	None
	IO$M_ATTNAST*	100	P1 = AST address
	IO$M_BUFFERED	1000	None
	IO$M_DIRECT	2000	None
	IO$M_TIMOUT	4000	P1 = seconds
IO$_WRITE{L,P,V}BLK*		{20,0B,30}	P1 = bufadr, P2 = bytcnt, [P5 = listener address]
	IO$M_EOI	80	
	IO$M_ENA_SRQ	100	
	IO$M_CONTROL	200	
	IO$M_STANDBY	400	
	IO$M_LEAVIS	800	

*These are standard VMS function codes and modifiers.

The SYS$QIO call requires that six arguments, P1 through P6, be specified. Arguments not used by a function code should be specified as null arguments. Null arguments are indicated by commas only. For example, in the following line of code, arguments P3 through P6 are null arguments:

```
status = sys$qio(0,chan,IO$_READLBLK,iosb,ast,ast_param,BUF1,3000,,,,)
```

3.3.1 IO$_AUXILIARY($) - Issue an Auxiliary Command

The IO$_AUXILIARY function is used to issue an auxiliary command. A detailed knowledge of the IEC/IEEE bus is required to understand the use of auxiliary commands.

For a complete description of the IEX11 auxiliary commands refer to Chapter 3 of the *IEU11-A/IEQ11-A User's Guide*.

Function code arguments and definitions for the IO$_AUXILIARY function are listed below.

Argument	VMS Usage	Type	Access	Passed by	Meaning
P1	Longword signed	Longword integer (signed)	Read Only	Value	low order byte contains auxiliary command

Many auxiliary commands require that bit seven of the command byte be set to enable the feature. To disable the feature, the auxiliary command is issued with bit seven of the command byte cleared.

The typical user will only use auxiliary commands to perform a DAC hold-off and to set devices to their remote state. Devices are set to remote by issuing the send remote enable (**sre**) auxiliary command with bit seven set and then addressing the devices as listener.

Table 3–5 Auxiliary Commands

	(Octal Code)		
Mnemonic	**Disable**	**Enable**	**Special Function**
swrst	000	200	Chip Reset *
dacr	001	201	Release DAC hold-off *
rhdf		002	Release RFD hold-off
hdfa	003	203	Hold-off on all data *
hdfe	004	204	Hold-off on EOI only *
nbaf		005	Set new byte available false
fget	006	206	Force group execute trigger *
rtl	007	207	Return to local *
feoi		010	Send EOI with next byte
lon	011	211	Listen only *
ton	012	212	Talk only *
gts		013	Go to standby
tca		014	Take control asynchronously
tcs		015	Take control synchronously
rpp	016	216	Request parallel poll *
sic	017	217	Send interface clear *
sre	020	220	Send remote enable *
rqc		021	Request control
rlc		022	Release control
dai	023	223	Disable all interrupts *
pts		024	Pass through next secondary
stdw	025	225	Short T1 settling time *
shdw	026	226	Shadow handshake *
vstdl	027	227	Very short T1 delay *
rsv2	030	230	Request service bit 2 *

*Enabled by issuing the auxiliary command with bit seven set and are disabled by issuing the auxiliary command with bit seven clear.

The driver frequently uses auxiliary commands while processing other SYS$QIO requests. For example, the IO$_PASSCONTROL function uses the release control (**rlc**) auxiliary command.

3.3.1.1 IO$_AUXILIARY($) I/O Status Block

No function specific information is returned in the IOSB. Refer to Section 3.2 for a description of the contents of the IOSB.

3.3.1.2 IO$_AUXILIARY($) Status Codes

The I/O status codes are returned in the low word of the first longword of the IOSB or by the SYS$QIO call. They are shown below.

Status Codes	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	Successful completion.
SS$_DEVREQERR	QIO	Request issued to a unit which is not initialized.
SS$_DEVACTIVE	QIO	Request issued to a unit which is in diagnostic mode.
SS$_TIMEOUT	IOSB	The request failed to complete within the specified time period.
SS$_ILLIOFUNC	IOSB	Request to issue an interface clear (sic) or to set remote enable (sre) by an IEX11 unit that is not the system controller.

3.3.2 IO$_COMMAND($) - Output One IEC/IEEE Command

The IO$_COMMAND function is used to issue one IEC/IEEE bus command. If the controller-in-charge only has to send one IEC/IEEE command, this request can save you some time. Rather than having to set up a command buffer and specify a byte count, you can simply specify the IEC/IEEE command directly in the P1 parameter.

For a discussion of the IEC/IEEE commands, refer to Section 3.3.3.

Function code arguments and definitions for the IO$_COMMAND function are listed below.

Argument	VMS Usage	Type	Access	Passed by	Meaning
P1	Longword signed	Longword integer (signed)	Read Only	Value	Low order byte contains command

3.3.2.1 IO$_COMMAND($) I/O Status Block

Figure 3–2 shows the contents of the I/O status block returned by the driver. The second word of the I/O status block will contain the number of commands transferred, which for this function is always 1.

Figure 3–2 IO$_COMMAND($) I/O Status Block

<table>
<tr><td colspan="4">number of commands transferred</td><td>return status</td></tr>
<tr><td>reserved</td><td>addressed state</td><td>reserved</td><td>contr state</td><td>IEC/IEEE status register</td></tr>
<tr><td>31</td><td>27</td><td>23</td><td>19</td><td>15 ... 0</td></tr>
</table>

Refer to Section 3.2 for a description of the contents of the IOSB which are not specific to this request.

3.3.2.2 IO$_COMMAND($) Status Codes

The I/O status codes are returned in the low word of the first longword of the IOSB or by the SYS$QIO call. They are shown below.

Status Codes	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	Successful completion.
SS$_DEVREQERR	QIO	Request issued to a unit which is not initialized.
SS$_DEVACTIVE	QIO	Request issued to a unit which is in diagnostic mode.
SS$_ILLIOFUNC	IOSB	The request was issued to a unit that is not controller-in-charge.
SS$_TIMEOUT	IOSB	The request failed to complete within the specified time period.

3.3.3 IO$_COMMANDS($) - Output a Buffer of IEC/IEEE Commands

The IO$_COMMANDS function is used to write a buffer of IEC/IEEE bus commands onto the IEC/IEEE bus.

Function code arguments and definitions for the IO$_COMMANDS function are listed below.

Argument	VMS Usage	Type	Access	Passed by	Meaning
P1	Longword unsigned	Longword integer (unsigned)	Read Only	Reference	Address of a byte array containing the commands
P2	Longword signed	Longword integer (signed)	Read Only	Value	Number of commands to be transferred

IEC/IEEE commands are divided into the following groups:

Table 3–6 IEC/IEEE Command Groups

IEC/IEEE Command	Group
0_8 - 17_8	Addressed Command Group (ACG)
20_8 - 37_8	Universal Command Group (UCG)
40_8 - 77_8	Listen Address Group (LAG)
100_8 - 137_8	Talk Address Group (TAG)
140_8 - 177_8	Secondary Command Group (SCG)

Since IEC/IEEE bus commands require that the ATN line be asserted, they can only be issued by the controller-in-charge when it is in the controller active state (CACS). If the IEX11 unit is controller-in-charge, but in the controller standby state (CSBS), the driver will automatically set the unit to CACS before transferring the commands. It will then reset the unit to CSBS after the transfer is complete.

Frequently, this request is used by an IEX11 controller-in-charge to address other devices on the IEC/IEEE bus as listeners or talkers. The IEC/IEEE command for addressing a device as a listener is 40_8 plus the IEC/IEEE bus address of that device. The IEC/IEEE command for addressing a device as a talker is 100_8 plus the IEC/IEEE bus address of that device.

Primary listener addresses are in the range 40_8 to 76_8. In order to address device number one as a listener, a value of 41_8 must be sent. A value of 77_8 is used to deaddress all listeners on the IEC/IEEE bus. Similarly, the primary talker addresses are in the range 140_8 to 1 36_8 and 137_8 is used to deaddress the talker on the bus.

The only way that an IEX11 can address itself as a listener or talker is by issuing the **lon** or **ton** auxiliary commands. However, if a command that would affect the addressed state of an IEX11 controller-in-charge is issued, the driver will automatically issue the correct auxiliary command.

For example, if an IEX11 unit with IEC/IEEE bus address 0 issues the IEC/IEEE command 40_8, the driver will issue the **ton** auxiliary command and also send the 100_8 on the bus. The my talk address (MTA) command is issued to deaddress any active talker on the bus.

3.3.3.1 IO$_COMMANDS($) I/O Status Block

Figure 3–3 shows the contents of the I/O status block returned by the driver. The second word of the I/O status block will contain the number of commands transferred.

Figure 3–3 IO$_COMMANDS($) I/O Status Block

number of commands transferred				return status
reserved	addressed state	reserved	contr state	IEC/IEEE status register
31	27	23	19	15 ... 0

Refer to Section 3.2 for a description of the contents of the IOSB which are not specific to this request.

3.3.3.2 IO$_COMMANDS($) Status Codes

The I/O status codes are returned in the low word of the first longword of the IOSB or by the SYS$QIO call. They are shown below.

Status Codes	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	Successful completion.
SS$_ACCVIO	QIO	No read access to specified buffer.
SS$_BADPARAM	QIO	Invalid byte count specified.
SS$_DEVACTIVE	QIO	Request issued to a unit which is in diagnostic mode.
SS$_DEVREQERR	QIO	Request issued to a unit which is not initialized.
SS$_ILLIOFUNC	IOSB	The request was issued to a unit that is not controller-in-charge.
SS$_TIMEOUT	IOSB	The request failed to complete within the specified time period.

3.3.4 IO$_GO_TO_CACS($) - Go To Controller Active State

The IO$_GO_TO_CACS function is used by an IEX11 unit that is controller-in-charge to change from controller standby state (CSBS) to controller active state (CACS). If the unit is already in CACS when the request is issued, no operation is performed. Since the driver will automatically try to go to the state required to process requests, IO$_GO_TO_CACS is not normally needed.

There are no function specific arguments or modifiers for the IO$_GO_TO_CACS function.

3.3.4.1 IO$_GO_TO_CACS($) I/O Status Block

No function specific information is returned in the IOSB. Refer to Section 3.2 for a description of the contents of the IOSB.

3.3.4.2 IO$_GO_TO_CACS($) Status Codes

The I/O status codes are returned in the low word of the first longword of the IOSB or by the SYS$QIO call. They are shown below.

Status Codes	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	Successful completion.
SS$_DEVREQERR	QIO	Request issued to a unit which is not initialized.
SS$_DEVACTIVE	QIO	Request issued to a unit which is in diagnostic mode.
SS$_ILLIOFUNC	IOSB	The request was issued to a unit that is not controller-in-charge.

3.3.5 IO$_GO_TO_CSBS($) - Go To Controller Standby State

The IO$_GO_TO_CSBS function is used by an IEX11 unit that is controller-in-charge to change from controller active state (CACS) to controller standby state (CSBS). If the unit is already in CSBS when the request is issued, no operation is performed.

There are no function specific arguments or modifiers for the IO$_GO_TO_CSBS function.

The two main uses for this request are:

- To improve performance when the controller-in-charge issues several read requests to receive a large buffer of data from a device.
- To permit other devices on the bus to transfer data among themselves.

If the controller-in-charge issues several read requests to receive on buffer from a talker, placing the controller in CSBS before the first read will improve performance. The reasons for this are presented in Section 3.3.11 which describes the IO$_READPBLK request.

The controller-in-charge must be placed into CSBS when it is not an active participant (either as a talker or a listener) in a data transfer on the IEC/IEEE bus. In this situation, the controller-in-charge would first address one device as talker and another as a listener. The controller would then issue the IO$_GO_TO_CSBS function to place itself into controller standby state, deasserting the attention line so that the talker could send data to the listener.

3.3.5.1 IO$_GO_TO_CSBS($) I/O Status Block

No function specific information is returned in the IOSB. Refer to Section 3.2 for a description of the contents of the IOSB.

3.3.5.2 IO$_GO_TO_CSBS($) Status Codes

The I/O status codes are returned in the low word of the first longword of the IOSB or by the SYS$QIO call. They are shown below.

Status Codes	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	Successful completion.
SS$_DEVREQERR	QIO	Request issued to a unit which is not initialized.
SS$_DEVACTIVE	QIO	Request issued to a unit which is in diagnostic mode.
SS$_ILLIOFUNC	IOSB	The request was issued to a unit that is not controller-in-charge.

3.3.6 IO$_INITIALIZE($) - Initialize Unit

The IO$_INITIALIZE function is used to initialize an IEX11 unit. It must be the first request issued to the unit by the first process to access the unit. Subsequent calls to IO$_INITIALIZE must be synchronized between processes using the unit so as not to result in the loss of data from outstanding requests. A request issued to a noninitialized unit will generate an SS$_DEVREQERR error. The bits set in the mask parameter (P1) determine how the unit will be initialized. Table 3–7 describes the meaning of the mask parameter bits.

Function code arguments and definitions for the IO$_INITIALIZE function are listed below.

Argument	VMS Usage	Type	Access	Passed by	Meaning
P1	Longword signed	Longword integer (signed)	Read Only	Value	Low order byte contains mask indicating initialization options
P2	Longword signed	Longword integer (signed)	Read Only	Value	IEC/IEEE bus address of IEX11 controller

If a unit is being initialized as controller-in-charge, then a mask value of 67_8 is normally specified. If the unit is not controller-in-charge, a mask of 43_8 is normally specified.

The P2 argument specifies the primary address of the unit. Valid primary addresses range from 0 to 36_8.

Note: All IEX11 units connected on a common bus must be initialized BEFORE any are utilized.

Table 3–7 Initialization Mask Bit Definitions

Bit	Action Taken
0	Issue master clear to interface
1	Send software reset
2	Set system controller bit (refer to Chapter 1)
3	Clear system controller bit (refer to Chapter 1)
4	Set controller to Controller Active State (CACS)
5	Set primary address

3.3.6.1 IO$_INITIALIZE($) I/O Status Block

No function specific information is returned in the IOSB. Refer to Section 3.2 for a description of the contents of the IOSB.

3.3.6.2 IO$_INITIALIZE($) Status Codes

The I/O status codes are returned in the low word of the first longword of the IOSB or by the SYS$QIO call. They are shown below.

Status Codes	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	Successful completion.
SS$_DEVACTIVE	QIO	Request issued to a unit which is in diagnostic mode.
SS$_ILLIOFUNC	IOSB	The mask parameter indicated that the unit should go to CACS, but the unit is not system controller.

3.3.7 IO$_LOADPARPOLL($) - Load Parallel Poll

The IO$_LOADPARPOLL function is used to load a status byte into the IEX11 unit's parallel poll register. This request is normally issued only to a unit that is not controller-in-charge.

Function code arguments and definitions for the IO$_LOADPARPOLL function are listed below.

Argument	VMS Usage	Type	Access	Passed by	Meaning
P1	Longword signed	Longword integer (signed)	Read Only	Value	Low order byte contains status byte

Normally, each device participating in a parallel poll is assigned a specific bit (IO$_PARPOLLCON. During a parallel poll, the controller-in-charge reads the OR of the parallel poll registers of all devices configured to participate in the parallel poll. The user's program should, therefore, issue an IO$_LOADPARPOLL whenever the state of the IEX11 unit which will participate in a parallel poll changes. The user has complete control in defining what constitutes a change in state for an IEX11.

3.3.7.1 IO$_LOADPARPOLL($) I/O Status Block

No function specific information is returned in the IOSB. Refer to Section 3.2 for a description of the contents of the IOSB.

3.3.7.2 IO$_LOADPARPOLL($) Status Codes

The I/O status codes are returned in the low word of the first longword of the IOSB or by the SYS$QIO call. They are shown below.

Status Codes	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	Successful completion.
SS$_DEVREQERR	QIO	Request issued to a unit which is not initialized.
SS$_DEVACTIVE	QIO	Request issued to a unit which is in diagnostic mode.

3.3.8 IO$_PAR_POLL($) - Perform a Parallel Poll

The IO$_PAR_POLL function causes the controller-in-charge to perform a parallel poll. The parallel poll allows the state of up to eight devices to be quickly and simultaneously determined. The result of the parallel poll is a status byte whose value is the OR of the parallel poll register of all devices configured to participate in the parallel poll.

There are no function specific arguments or modifiers for the IO$_PAR_POLL function.

Before a parallel poll can be performed, the controller-in-charge must configure the devices that are to participate in the parallel poll by issuing the IO$_PARPOLLCON request. The IO$_PARPOLLCON request assigns each device one (or more) bits through which to indicate its state.

The driver performs the following steps when conducting a parallel poll:

1. Controller is set to CACS if not already in an active state.
2. The **rpp** enable auxiliary command is issued.
3. The driver waits approximately 150 microseconds for devices to respond to the parallel poll. Devices respond by simultaneously placing the values in their parallel poll registers on the IEC/IEEE bus.
4. The resulting status byte is read from the command pass through register.
5. The driver leaves the parallel poll state by issuing the **rpp** disable auxiliary command.
6. Restores the controller state.

3.3.8.1 IO$_PAR_POLL($) I/O Status Block

Figure 3–4 shows the contents of the I/O status block returned by the driver. The second word of the I/O status block will contain the OR of the parallel poll registers of all devices participating in the parallel poll.

Figure 3–4 IO$_PAR_POLL($) I/O Status Block

OR of parallel poll registers for all configured devices				return status
reserved	addressed state	reserved	contr state	IEC/IEEE status register
31	27	23	19	15 … 0

Refer to Section 3.2 for a description of the contents of the IOSB which are not specific to this request.

3.3.8.2 IO$_PAR_POLL($) Status Codes

The I/O status codes are returned in the low word of the first longword of the IOSB or by the SYS$QIO call. They are shown below.

Status Codes	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	Successful completion.
SS$_DEVREQERR	QIO	Request issued to a unit which is not initialized.
SS$_DEVACTIVE	QIO	Request issued to a unit which is in diagnostic mode.
SS$_ILLIOFUNC	IOSB	The request was issued to a unit that is not controller-in-charge.

3.3.9 IO$_PARPOLLCON($) - Parallel Poll Configure

The IO$_PARPOLLCON function is used by an IEX11 unit that is controller-in-charge to configure devices on the IEC/IEEE bus to respond to a parallel poll. It associates specific bits in the parallel poll register to devices, and instructs the devices as to how they are to indicate their states.

Function code arguments and definitions for the IO$_PARPOLLCON function are listed below.

Argument	VMS Usage	Type	Access	Passed by	Meaning
P1	Longword unsigned	Longword integer (unsigned)	Read Only	Reference	Address of byte array containing parallel poll configuration data
P2	Longword signed	Longword integer (signed)	Read Only	Value	Length of configuration buffer in bytes

The parallel poll provides a mechanism for reading the status of up to eight devices simultaneously. Each device is assigned one or more bits through which to indicate its state. The devices must be capable of participating in a parallel poll and must update their status registers to correctly reflect their states.

The configuration buffer contains pairs of listener addresses and configure bytes, as shown in Figure 3–5.

Figure 3–5 Parallel Poll Configuration Buffer

byte 0	byte 1	byte 2	byte 3	...
listener address 1	configure byte 1	listener address 2	configure byte 2	... ...

Each configure byte has the following format:

Bits	Meaning
0 - 2	Specifies the bit (0 - 7) that the device will respond to during a parallel poll.
3	Specifies the state (0 or 1) of the bit in the parallel poll register if the state of the device is true.

IO$_PARPOLLCON causes the following sequence of events to be performed:

1. Controller is set to CACS if not already in an active state, and the previous state is preserved.
2. The UNL IEC/IEEE command is transmitted.
3. The first specified device from the P1 buffer is checked to be a valid listener address, then it is transmitted.
4. The addressed command parallel poll configure (PPC) is transmitted to enable the device addressed as listener.
5. The configure byte for that device is transmitted. Before sending the configure byte, the driver sets bits five and six to ensure that the byte has the proper format.
6. Unlisten all devices.
7. If more devices exist in the P1 buffer, the driver checks for a valid listener address, transmits it, and continues at step 4.
8. Controller state is restored.

A maximum of eight devices may be configured to respond to a parallel poll. The controller-in-charge may command a device to cease responding to a parallel poll, by sending it a parallel poll disable (PPD) IEC/IEEE command for that address from the secondary command group subset. The subset of devices participating in the parallel poll can, therefore, be varied by using the unconfigure command to remove devices and the parallel poll request to add devices.

This request may only be issued to a unit which is controller-in-charge.

3.3.9.1 IO$_PARPOLLCON($) I/O Status Block

Figure 3–6 shows the contents of the I/O status block returned by the driver. The second word of the I/O status block will contain the number of devices configured.

Figure 3–6 IO$_PARPOLLCON($) I/O Status Block

number of devices configured				return status
reserved	addressed state	reserved	contr state	IEC/IEEE status register
31	27	23	19	15 — 0

Refer to Section 3.2 for a description of the contents of the IOSB which are not specific to this request.

3.3.9.2 IO$_PARPOLLCON($) Status Codes

The I/O status codes are returned in the low word of the first longword of the IOSB or by the SYS$QIO call. They are shown below.

Status Codes	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	Successful completion.
SS$_DEVACTIVE	QIO	Request issued to a unit which is in diagnostic mode.
SS$_ILLIOFUNC	IOSB	The request was issued to a unit that is not controller-in-charge.
SS$_IVADDR	IOSB	The buffer specified by P1 contains an invalid listener address.
SS$_TIMEOUT	IOSB	The request failed to complete within the specified time period.

3.3.10 IO$_PASSCONTROL($) - Pass Control

The IO$_PASSCONTROL function is used by an IEX11 unit that is controller-in-charge to pass control to another device on the IEC/IEEE bus. The P1 parameter specifies the talker address of the device that is to receive control.

Function code arguments and definitions for the IO$_PASSCONTROL function are listed below.

Argument	VMS Usage	Type	Access	Passed by	Meaning
P1	Longword signed	Longword integer (signed)	Read Only	Value	Talker address of device to receive control

If control is passed from one IEX11 unit to another IEX11 unit, both units will be placed in CACS if the request times out. This situation will occur if the received control event is enabled and the unit receiving control does not issue a **dacr** within the timeout period. The driver will place the unit receiving control in CACS, but the timeout will abort the IO$_PASSCONTROL request before the **rlc** auxiliary command has been executed.

This request may only be issued to a unit which is controller-in-charge.

3.3.10.1 IO$_PASSCONTROL($) I/O Status Block

No function specific information is returned in the IOSB. Refer to Section 3.2 for a description of the contents of the IOSB.

3.3.10.2 IO$_PASSCONTROL($) Status Codes

The I/O status codes are returned in the low word of the first longword of the IOSB or by the SYS$QIO call. They are shown below.

Status Codes	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	Successful completion.
SS$_DEVACTIVE	QIO	Request issued to a unit which is in diagnostic mode.
SS$_DEVREQERR	QIO	Request issued to a unit which is not initialized.
SS$_ILLIOFUNC	IOSB	The request was issued to a unit that is not controller-in-charge.
SS$_IVADDR	IOSB	The buffer specified by P1 contains an invalid talker address
SS$_TIMEOUT	IOSB	The request failed to complete within the specified time period.

3.3.11 IO$_READ{L,P,V}BLK - Read Data

The IO$_READLBLK, IO$_READPBLK, and IO$_READVBLK functions are used to read data in DMA mode from the device on the IEC/IEEE bus that is addressed as talker. All three requests are processed in the same manner by the driver.

Function code arguments, modifiers, and definitions for the IO$_READxBLK functions are listed below.

Argument	VMS Usage	Type	Access	Passed by	Meaning
P1	Longword unsigned	Longword integer (unsigned)	Read Only	Reference	Address of byte array into which the data will be read
P2	Longword signed	Longword integer (signed)	Read Only	Value	Number of bytes to be received; range: 1...65535
P3	Longword signed	Longword integer (signed)	Read Only	Value	Low order byte contains the match character (optional)
P4	Longword unsigned	Longword integer (unsigned)	Read Only	Value	Match character count (optional)
P5	Longword unsigned	Longword integer (unsigned)	Read Only	Value	Talker address (optional)

The P1 and P2 parameters specify the address of the data buffer and the number of bytes to be read. The number of bytes actually read is returned in the second word of the I/O status buffer.

If P4 is non-zero, the driver enables match character detection. When this option is enabled, data transfers will terminate when the value in the low byte of P3 is consecutively received the number of times specified by P4. The match character count (P4) must be in the range 1 to 63_{10}. If P4 is zero, match character detection is not enabled. Refer to the modifier description section for a description of the restriction when the IO$M_TCS modifier is used with match character termination enabled.

If the IEX11 unit issuing the read request is controller-in-charge, and P5 contains a valid **talker address**, then the IEX11 unit will address the device specified by P5 as talker before reading the data. This feature is useful for applications where timing is critical, since it saves the overhead of an extra $QIO to address the device as talker. The device that was addressed as talker remains in the talker state after the read request terminates.

The IEX11 (or any device) must be addressed as listener before it can receive data. If the unit issuing the read request is in CACS and not addressed as listener when the request is issued, the driver will automatically issue a **lon** auxiliary command to address the unit as listener. The driver deaddresses the unit after the transfer has completed.

If the IEX11 unit is in CACS when the read request is issued, the driver will automatically set the unit to CSBS so that the data transfer can occur. It will reset the unit to CACS after the data transfer, unless the IO$M_LEAVIS modifier was specified.

If the unit is not controller-in-charge and not addressed as listener when the request is issued, the driver will wait for the unit to be addressed as listener before initiating the data transfer.

The read request effectively blocks the IEX11 unit. When the unit is addressed as listener after a read request has been issued, the driver does not generate an *addressed as listener* event. However, the driver will automatically issue the **dacr** since the user's process would not be able to issue it.

Note: The automatic features of the IO$_READxBLK function (detecting a talker address in P5, issuing the *lon* auxiliary command, and placing the unit in CSBS) minimize the number of required QIOs. However, these features should NOT BE USED in applications where several read requests are required to read a single buffer of data from a device. The proper procedure for such applications is to address the transmitting device as talker and the receiving device as listener then put the controller in CSBS before issuing the read requests.

Data can be read through either a direct or buffered data path. The IO$_SETMODE function can be used to select the method best suited for the application. The selection of a data path should be considered fine tuning since it only affects the speed of the transfer. Direct data path is the default.

Read requests may be terminated in one of three ways:

- Byte count overflow
- EOI termination
- Match character detection

Byte count overflow occurs after the number of bytes specified by the P2 parameter has been read. If the read will be terminated by match character or EOI, then P2 specifies the maximum number of bytes that can be read. Normally, P2 is the length of the buffer (in bytes) pointed to by P1.

EOI termination occurs when a byte is read with the EOI line asserted.

Modifier	Meaning
IO$M_TCS	Used to instruct the driver to automatically issue a **tcs** auxiliary command prior to reading the last byte of the transfer. This ensures that the unit will enter CACS before the talker can send another byte. This modifier can sometimes prevent data loss when several read requests are used to receive one buffer from a fast device. The following example using the IEX exerciser shows a write request being interrupted by as serial poll. This example will not work correctly if the IO$M_TCS modifier is removed:

```
INI 0 67 0
INI 1 43 1
CMD 0 101
WLB 1 8.
RLB!IO$M_TCS 0 4
WFT 0
SPO!IO$M_ALLDEVICES 0 101
CMD 0 101
RLB 0 4
WFT 0
```

This modifier may only be used for a unit which is CIC.

Note: Use of the IO$M_TCS modifier when match character (P3) is specified is restricted to the single case where the byte count (P2) is exactly equal to the number of bytes actually read in at the time the match count (P4) criteria is met. The driver will not take control if IO$M_TCS is specified and the number of bytes specified in P2 is greater then the number of bytes actually read. This is a permanent restriction in the IEX-VMS-Driver due to the implementation of match character in the IEX11 hardware.

Modifier	Meaning
IO$M_ENA_SRQ	Used to allow service requests to terminate the data transfer. Logic in the IEX11 hardware automatically terminates DMA transfers when an interrupt occurs. The IEX11 driver, therefore, clears the interrupt mask before starting DMA transfers, thus preventing interrupts from terminating the data transfer. This modifier causes the driver to set the service request bit in the interrupt mask register prior to initiating the DMA transfer, thereby allowing service requests to interrupt the data transfer. If a service request interrupts a data transfer, the read request will be returned with the SS$_DEVREQERR status code.
IO$M_CONTROL	Used to leave the controller-in-charge in CACS when the transfer completes. This modifier may only be used for a unit which is CIC.
IO$M_STANDBY	Used to leave the controller-in-charge in CSBS when the transfer completes. This modifier may only be used for a unit which is CIC.

Modifier	Meaning
IO$M_LEAVIS	Used to leave the controller-in-charge in the state which was required to process the request. This modifier may only be used for a unit which is CIC.

3.3.11.1 IO$_READ{L,P,V}BLK I/O Status Block

Figure 3–7 shows the contents of the I/O status block returned by the driver. The second word of the I/O status block will contain the number of bytes written.

Figure 3–7 IO$_READ{L,P,V}BLK I/O Status Block

number of bytes read*				return status	
reserved	addressed state	term type	contr state	IEC/IEEE status register	
31	27	23	19	15	0

* For SS$_DEVICEFULL error, this word contains the previous talker address, which sent the byte.

The type of termination is returned in bits 20-23 of the second longword of the I/O status buffer. The following values may be returned:

0 - Byte count overflow

1 - EOI termination (overwrites byte count overflow and match character detection if also true)

2 - Match character detection (overwrites byte count overflow if also true)

Refer to Section 3.2 for a description of the contents of the IOSB which are not specific to this request.

3.3.11.2 IO$_READ{L,P,V}BLK Status Codes

The I/O status codes are returned in the low word of the first longword of the IOSB or by the SYS$QIO call. They are shown below.

Status Codes	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	Successful completion.
SS$_DATACHECK	IOSB	Request was terminated as a result of an IFC or ERR interrupt.
SS$_DEVACTIVE	QIO	Request issued to a unit which is in diagnostic mode.
SS$_DEVICEFULL	IOSB	A byte was transferred to the IEX11 by a device other than the current talker. A QIO READ of one byte must be issued to allow subsequent reads to function.
SS$_DEVREQERR	QIO	Request issued to a unit which is not initialized.
SS$_DEVREQERR	IOSB	Request was terminated as a result of a service request (IO$M_ENA_SRQ modifier required).
SS$_ILLIOFUNC	IOSB	The request was issued to a unit that is not controller-in-charge.
SS$_IVADDR	IOSB	The buffer specified by P5 contains an invalid talker address.
SS$_TIMEOUT	IOSB	The request failed to complete within the specified time period.

3.3.12 IO$_REC_EVENT($) - Recognize Event

The IO$_REC_EVENT function is used to receive notification of events that have been previously enabled through the IO$_SETEVENT request (Section 3.3.16). Information on the event that occurred is returned in the I/O status buffer.

There are no function specific arguments or modifiers for the IO$_REC_EVENT function.

The IEX11 hardware terminates DMA transfers when an interrupt occurs. The IEX-VMS-Driver, therefore, disables event recognition before a data transfer is started and reenables it after the transfer completes.

When an event (other than service request, remote/local change, or interface clear) that has been enabled by the IO$_SETEVENT function occurs, the IEC/IEEE bus becomes locked. The bus will not accept commands or transfer data. The bus is locked to prevent multiple occurrences of events.

The bus is unlocked by issuing a **dacr** auxiliary command. The driver will automatically issue this auxiliary command when the next request to the unit is issued. There are two exceptions to this rule. The driver will not automatically issue a **dacr** auxiliary command if the request is IO$_SENSEMODE, or if the request is for a **dacr** auxiliary command.

The only case in which your program should not let the driver automatically perform the **dacr**, is when an extended address has been received. Enabling **dacr** indicates that the unit has accepted the extended address. Disabling **dacr** indicates that the extended address has been rejected. If a disable **dacr** is issued, the IEX11 unit will exit the addressed state.

Only events which have been enabled by the IO$_SETEVENT function require **dacr** enabling. To maximize performance, the application should only enable the recognition of events to which it must respond.

If an event occurs in the period between the **dacr** and the next issuance of an IO$_REC_EVENT request, the event will be saved internally in the driver. The pending event will be returned immediately when the next IO$_REC_EVENT request is issued. If no event is pending when an IO$_REC_EVENT request is issued, the driver will wait until the next event occurs or the specified timeout period expires. The driver sets the IEX11 unit to not busy for this period so that it can process other I/O requests. This feature is especially important for an IEX11 unit that is controller-in-charge. The CIC can have an IO$_REC_EVENT outstanding to detect service requests at the same time that it is participating in data transfers with other devices on the IEC/IEEE bus.

A *service request* event is not generated if a device on the IEC/IEEE bus requests service before the IO$_INITIALIZE function initializes the controller-in-charge. Such a *service request* event would be lost because of the reset functions performed by IO$_INITIALIZE. To check if an SRQ was issued before the controller-in-charge was initialized, inspect the SRQ bit of the IEC/IEEE status register, returned in the third word of the I/O status buffer. If the SRQ bit is set, a *service request* is pending, and a serial poll should be performed.

An alternate way to recognize events is through the use of attention ASTs. An advantage of using attention ASTs to recognize events is that the IEX11 unit is not blocked and can be used to process other requests. However, programming attention AST routines is more complicated than programming the IO$_REC_EVENT request. Refer to Section 3.3.20 for additional information on the use of attention ASTs.

Function code modifiers and definitions for the IO$_REC_EVENT function are listed below.

Modifier	Meaning
IO$M_NOWAIT	Used to instruct the driver not to wait for an event to occur. If an event has occurred, the event information is returned. This modifier allows your task to poll the driver to determine if an event has occurred. The request returns the SS$_ MSGNOTFND status code if not event has occurred.

3.3.12.1 IO$_REC_EVENT I/O Status Block

Figure 3–8 shows the contents of the I/O status block returned by the driver. The second word of the I/O status block will contain the number of devices polled.

Figure 3–8 IO$_REC_EVENT I/O Status Block

<table>
<tr><td colspan="2">event information</td><td colspan="2">event code</td><td>return status</td></tr>
<tr><td>reserved</td><td>addressed state</td><td>reserved</td><td>contr state</td><td>IEC/IEEE status register</td></tr>
<tr><td>31</td><td>27</td><td>23</td><td>19</td><td>15 … 0</td></tr>
</table>

The **event code** byte indicates which event occurred. The following values may be returned:

Value	Event
1	Service request
2	Addressed as listener
3	Addressed as talker
4	Deaddressed
5	Addressed as extended listener
6	Addressed as extended talker
7	Device clear
8	Group execute trigger
9	Remote/local change
10	Received control
11	Parallel poll configure
12	Parallel poll unconfigure
13	Interface clear

The **event information** byte contains the following information:

- For *addressed as extended listener* and *addressed as extended talker* events, the secondary address is returned in this byte.
- For *remote/local change*, this byte contains 0 if the controller is in local mode and 1 if it is in remote mode.
- For *parallel poll configure* and *parallel poll unconfigure* events, this byte contains the parallel poll byte.
- For all other events, this byte contains 0.

Refer to Section 3.2 for a description of the contents of the IOSB which are not specific to this request.

3.3.12.2 IO$_REC_EVENT Status Codes

The I/O status codes are returned in the low word of the first longword of the IOSB or by the SYS$QIO call. They are shown below.

Status Codes	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	Successful completion.
SS$_DEVACTIVE	QIO	Request issued to a unit which is in diagnostic mode.
SS$_DEVREQERR	QIO	Request issued to a unit which is not initialized.
SS$_ILLIOFUNC	IOSB	No events are enabled on the unit for current process.
SS$_MSGNOTFND	IOSB	The request was issued with the IO$M_NOWAIT modifier and no event had occurred.
SS$_TIMEOUT	IOSB	The request was issued without the IO$M_NOWAIT modifier and no event occurred.

3.3.13 IO$_SENSEMODE - Obtain Information About a Unit

The IO$_SENSEMODE function is used to obtain information on the state of an IEX11 unit. This request is processed by the preprocessing logic of the driver and is not placed in the request queue. It will complete immediately even if the unit is busy processing another request. The IO$_SENSEMODE function is, therefore, useful for determining the state of the unit when another request is hung.

To place an IO$_SENSEMODE request in the request queue, specify a dummy modifier. The dummy modifier may be any function modifier for a $QIO request. The request will then be queued to the driver in the normal manner.

There are no function specific arguments or modifiers for the IO$_SENSEMODE function.

3.3.13.1 IO$_SENSEMODE I/O Status Block

Information on the status of the unit is returned in the second longword of the I/O status buffer. Refer to Section 3.2 for a description of the contents of the IOSB.

3.3.13.2 IO$_SENSEMODE Status Codes

The I/O status codes are returned in the low word of the first longword of the IOSB or by the SYS$QIO call. They are shown below.

Status Codes	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	Successful completion.
SS$_DEVACTIVE	QIO	Request issued to a unit which is in diagnostic mode.
SS$_DEVREQERR	QIO	Request issued to a unit which is not initialized.

3.3.14 IO$_SER_POLL($) - Perform a Serial Poll

The IO$_SER_POLL function is used to perform a serial poll of selected devices on the IEC/IEEE bus to determine which devices require service.

Function code arguments, modifiers, and definitions for the IO$_SER_POLL function are listed below.

Argument	VMS Usage	Type	Access	Passed by	Meaning
P1	Longword unsigned	Longword integer (unsigned)	Read Only	Reference	Address of byte array containing the primary and secondary address of each device to be polled
P2	Longword signed	Longword integer (signed)	Read Only	Value	Length of address buffer in bytes
P3	Longword unsigned	Longword integer (unsigned)	Read Only	Reference	Address of byte array which will contain status bytes from the devices which have been polled

The P1 parameter specifies the address of the buffer containing talker addresses of the devices to be addressed in the serial poll. The P2 parameter specifies the length of the P1 buffer in bytes. The P1 buffer may contain a combination of primary and secondary addresses. For example, a typical buffer might contain the following:

Figure 3–9 Serial Poll Buffer

byte 0	byte 1	byte 2	byte 3	byte 4	...
talker address 1	talker address 2	extended address for talker address 2	talker address 3	talker address 4	... ...

When addressing a device in a serial poll, the driver first sends the primary talker address. It then examines the next address in the buffer. If this address is a valid secondary address (140_8 to 176_8), then it is also sent.

Each device addressed in the serial poll returns a status byte. If bit six (**rsv1**) of the status byte is set, the device currently addressed as talker had issued a service request. The driver stores these bytes in the buffer pointed to by P3. You must ensure that the P3 buffer is large enough to hold the status bytes of all devices to be polled.

Modifier	Meaning
IO$M_ALLDEVICES	Used to instruct the driver to poll all devices in the P1 buffer, regardless of the state of the SRQ line. Normally, the driver tests the SRQ line before each device is addressed in the serial poll. If the SRQ line is not asserted, the IO$_SER_POLL function is terminated.

The driver performs a number of operations when conducting a serial poll. Normally, the user need not be concerned with how the driver performs a serial poll. See the following steps:

1 The UNL command is issued to deactivate all listeners.

2 The SPE command is issued.

3 The IEX11 unit is set to listener by means of the **lon** auxiliary command.

4 If the IO$M_ALLDEVICES modifier is not specified, the SRQ bit in the IEX11 status register is tested. If the bit is not set, control resumes at step 14.

5 The next talker address is retrieved from the buffer and sent as an IEC/IEEE command. If the next byte in the buffer is an extended address, then this byte is also sent.

6 The ATN line is released by issuing a **gts** auxiliary command.

7 The device addressed as talker sends its status byte to the controller causing a **BI** (byte in) interrupt.

8 The driver gains control using the **tca** auxiliary command. This asserts the ATN line and puts the controller in CACS.

9 The driver sends UNT command to prevent multiple status bytes from being sent.

10 The driver reads the status byte from the Data In Register.

11 The status byte is transferred to the P3 buffer.

12 The count of the number of devices addressed in the serial poll is incremented.

13 If there are more addresses in the P1 buffer, control continues at step 4.

14 The serial poll is complete. The SPD command is issued.

15 The **lon** auxiliary command is issued to deaddress the controller.

It is possible for a fast device to send the first byte of its next transfer to the IEX11 while the system is still handling the last byte of the current transfer. In this case, the IEX11 will have a byte present when the serial poll command is issued. Since the serial poll status byte cannot be transferred, the serial poll request is aborted with a SS$_DEVICEFULL error. To clear the byte in the IEX11 Data In Register, the user must issue a one byte read, or issue a valid talker address command followed by a UNT command. Any QIO request which times out (returns SS$_TIMEOUT) also clears this byte.

If the device which issued the service request is not specified in the P1 buffer, or if a device which has been polled issues another service request before the serial poll completes, the bus will be locked until another serial poll is performed. To check if an SRQ is still outstanding at the completion of the serial poll, inspect the SRQ bit of the IEC/IEEE status register (returned in the third word of the I/O status buffer). If the SRQ bit is set, as service request is pending, and a serial poll should be performed.

This request may only be issued to a unit which is controller-in-charge.

3.3.14.1 IO$_SER_POLL I/O Status Block

Figure 3–10 shows the contents of the I/O status block returned by the driver. The second word of the I/O status block will contain the number of devices polled.

Figure 3–10 IO$_SER_POLL I/O Status Block

number of devices polled*				return status
reserved	addressed state	reserved	contr state	IEC/IEEE status register
31	27	23	19	15 … 0

* For SS$_DEVICEFULL error, this word contains the previous talker address, which sent the byte.

3.3.14.2 IO$_SER_POLL Status Codes

The I/O status codes are returned in the low word of the first longword of the IOSB or by the SYS$QIO call. They are shown below.

Status Codes	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	Successful completion.
SS$_DEVACTIVE	QIO	Request issued to a unit which is in diagnostic mode.
SS$_DEVICEFULL	IOSB	A byte was transferred to the IEX11 by a device other than the current talker. A QIO READ of one byte must be issued prior to issuing the serial poll.
SS$_DEVREQERR	QIO	Request issued to a unit which is not initialized.
SS$_ILLIOFUNC	IOSB	The request was issued to a unit that is not controller-in-charge.
SS$_IVADDR	IOSB	The buffer specified by P1 contains an invalid talker address.
SS$_TIMEOUT	IOSB	The request failed to complete within the specified time period.

3.3.15 IO$_SERVICE($) - Request Service

The IO$_SERVICE function can be used by an IEX11 unit that is not controller-in-charge to issue a service request. When the unit which issued the IO$_SERVICE function is addressed as talker in the subsequent serial poll, the status byte specified in the P1 parameter is transferred to the controller-in-charge.

Function code arguments and definitions for the IO$_SERVICE function are listed below.

Argument	VMS Usage	Type	Access	Passed by	Meaning
P1	Longword signed	Longword integer (signed)	Read Only	Value	Low order byte contains service request status byte
P2	Longword signed	Longword integer (signed)	Read Only	Value	Extended or secondary address

To generate a service request (assert the SRQ line), bit six (**rsv1**) of the service request status byte must be set. The other bits of this byte may be used to indicate the type of service requested. The driver will load the status byte into the serial poll register and wait to be addressed in a serial poll. The IO$_SERVICE function will not complete until the unit has transferred its status byte to the controller-in-charge by means of a serial poll or the specified timeout period has elapsed.

If this request is issued with **rsv1** of the service request status byte cleared, the driver will load the status byte into the serial poll register without generating a service request. The IO$_SERVICE function will then complete without waiting for a serial poll.

The optional P2 parameter can be used to specify an extended or secondary address. If P2 contains a valid secondary address, the driver will require the unit to receive both the specified primary and secondary addresses before it will respond to a serial poll.

This request may only be issued to a unit which is **not** controller-in-charge.

3.3.15.1 IO$_SERVICE I/O Status Block

No function specific information is returned in the IOSB. Refer to Section 3.2 for a description of the contents of the IOSB.

3.3.15.2 IO$_SERVICE Status Codes

The I/O status codes are returned in the low word of the first longword of the IOSB or by the SYS$QIO call. They are shown below.

Status Codes	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	Successful completion.
SS$_DEVACTIVE	QIO	Request issued to a unit which is in diagnostic mode.
SS$_DEVREQERR	QIO	Request issued to a unit which is not initialized.
SS$_IVADDR	IOSB	The buffer specified by P2 contains an invalid secondary address.
SS$_TIMEOUT	IOSB	The request failed to complete within the specified time period.

3.3.16 IO$_SETEVENT($) - Enable Event Recognition

The IO$_SETEVENT function is used to specify the events for which recognition is desired.

Function code arguments and definitions for the IO$_SETEVENT function are listed below.

Argument	VMS Usage	Type	Access	Passed by	Meaning
P1	Longword signed	Longword integer (signed)	Read Only	Value	Mask representing the events to enable

Events are selected by settings in the event mask (P1) parameter. Table 3–8 shows the definition of each bit in the event mask.

Table 3–8 Event Mask Bit Definitions

Bit	Event
none	Disable event recognition
0	Event on *service request*
1	Event on *addressed as listener*
2	Event on *addressed as talker*
3	Event on *deaddressed*
4	Event on *addressed as extended listener*
5	Event on *addressed as extended talker*
6	Event on *device clear*
7	Event on *group execute trigger*
8	Event on *remote/local change*
9	Event on *received control*
10	Event on *parallel poll configure*
11	Event on *parallel poll unconfigure*
12	Event on *interface clear*

The actual occurrence of events is detected through the IO$_REC_EVENT function or through the use of attention ASTs. However, events which have not been enabled by the IO$_SETEVENT function will not be returned by either mechanism.

For example, suppose an IEX11 unit that is not controller-in-charge is required to read data from a controller and send data to it in nonsequential order for a period of time. The IEX11 could be programmed to perform these operations in the following manner:

1. Enable recognition for the *addressed as talker* and *addressed as listener* events by issuing an IO$_SETEVENT request with the mask parameter equal to 6 (bits one and two set).
2. Issue an IO$_REC_EVENT request to wait for an event.
3. If the IO$_REC_EVENT returns *addressed as talker* event, transmit data. If the *addressed as listener* event is returned, receive data.
4. Issue another IO$_REC_EVENT to wait for another event.

Recognition of events specified in the IO$_SETEVENT request remains enabled until a subsequent IO$_SETEVENT request is issued.

Note: Before exiting, the process should disable event recognition by issuing an IO$_SETEVENT request with a mask parameter equal to 0. Failure to disable events will result in reduced system performance as the driver will continue to be interrupted by events on the bus and proceed to dismiss each one.

All events which are enabled by IO$_SETEVENT except service request, remote/local change, and interface clear, lock the IEC/IEEE bus when they occur. The bus remains locked until released through the **dacr** auxiliary command. Therefore, if the recognition of an event is enabled, the user's program must be able to detect the event and release the bus. Otherwise, traffic on the bus will hang.

A *service request* event is not generated if a device on the IEC/IEEE bus requests service before the IO$_INITIALIZE function initializes the controller-in-charge. Such a *service request* event would be lost because of the reset functions performed by IO$_INITIALIZE. To check if an SRQ was issued before the controller-in-charge was initialized, inspect the SRQ bit of the IEC/IEEE status register, returned in the third word of the I/O status buffer. If the SRQ bit is set, a *service request* is pending, and a serial poll should be performed.

Normally, service request events are only of interest to the controller-in-charge. With the exception of the interface clear, all other events are generally of interest only to IEX11 units that are not controller-in-charge. An interface clear may be of interest to any unit, regardless of its controller state.

The IEX11 interface can recognize either primary or extended (secondary) addressing, but not both at the same time. If recognition of extended addressing is enabled, the interface will no longer detect primary addressing. The user, therefore, cannot simply set all event mask bits (enable recognition of everything) since this enables both primary and secondary addressing. However, recognition of one of the addressing modes and all non-address events can be enabled.

When the controller-in-charge sends a selected device clear (SDC) or a group execute trigger (GET) command, it first addresses the selected devices as listener. If the selected *device clear* event or *group execute trigger* event is sent to an IEX11 unit that has enabled *addressed as listener* events, an *addressed as listener* event will be generated before the GET or SDC occurs. Thus, the event *addressed as listener* does not necessarily mean that an IEX11 should prepare to read data. Your program must wait for a *device clear* event, a *group execute trigger* event, or the attention (ATN) line to be deasserted prior to determining what action is appropriate.

3.3.16.1 IO$_SETEVENT I/O Status Block

No function specific information is returned in the IOSB. Refer to Section 3.2 for a description of the contents of the IOSB.

3.3.16.2 IO$_SETEVENT Status Codes

The I/O status codes are returned in the low word of the first longword of the IOSB or by the SYS$QIO call. They are shown below.

Status Codes	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	Successful completion.
SS$_DEVACTIVE	QIO	Request issued to a unit which is in diagnostic mode.
SS$_DEVREQERR	QIO	Request issued to a unit which is not initialized.

3.3.17 IO$_SETMODE - Set Mode of Operation

The IO$_SETMODE function is used to set miscellaneous parameters in the driver. Function modifiers are used to specify the parameter to be set.

Function code arguments, modifiers, and definitions for the IO$_SETMODE function are listed below.

Argument	VMS Usage	Type	Access	Passed by	Meaning
P1	Longword signed	Longword integer (signed)	Read Only	Value	Timeout duration in seconds, range: 2...32767 (only applicable if IO$M_TIMOUT modifier is specified)
P1	Longword unsigned	Longword integer (unsigned)	Read Only	Reference	Address of AST routine (only applicable if IO$M_ATTNAST modifier is specified)

Modifier	Meaning
IO$M_ATTNAST	Used to enable or disable attention ASTs when an event is detected. The P1 parameter specifies the starting address of the AST routine to which control will be passed when an event enabled by IO$_SETEVENT occurs. If P1 equals 0, attention ASTs are disabled. Refer to Section 3.3.20 for more information on using attention ASTs.
IO$M_BUFFERED	Used to specify that data is to be transferred by means of a buffered data path. Data errors will result if this request is used on a VAX-11/780 system.
IO$M_DIRECT	Used to specify that data is to be transferred using a direct data path. This is the default transfer mode.
IO$M_TIMOUT	Used to alter the length of time the driver will wait for an expected interrupt. If the interrupt does not occur within this period, the request is returned with an SS$_TIMEOUT status. Note that many requests to the IEX-VMS driver generate a series of interrupts and that the timeout period is reset after each interrupt, thereby having a cumulative effect. The default timeout period is 20 seconds. The IO$_INITIALIZE request resets the timeout period to this value whenever it is called.

3.3.17.1 IO$_SETMODE I/O Status Block

No function specific information is returned in the IOSB. Refer to Section 3.2 for a description of the contents of the IOSB.

3.3.17.2 IO$_SETMODE Status Codes

The I/O status codes are returned in the low word of the first longword of the IOSB or by the SYS$QIO call. They are shown below.

Status Codes	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	Successful completion.
SS$_DEVACTIVE	QIO	Request issued to a unit which is in diagnostic mode.
SS$_DEVREQERR	QIO	Request issued to a unit which is not initialized.
SS$_BADPARAM	IOSB	P1 contains an invalid timeout value.

3.3.18 IO$_WRITE{L,P,V}BLK - Write Data

The IO$_WRITELBLK, IO$_WRITEPBLK, and IO$_WRITEVBLK functions are used to transfer data in DMA mode to all devices on the IEC/IEEE bus addressed as listener. All three requests are processed in the same manner by the driver.

Function code arguments, modifiers, and definitions for the IO$_WRITExBLK functions are listed below.

Argument	VMS Usage	Type	Access	Passed by	Meaning
P1	Longword unsigned	Longword integer (unsigned)	Read Only	Reference	Address of byte array containing the values to be written
P2	Longword signed	Longword integer (signed)	Read Only	Value	Number of bytes to be transferred; range: 1...65535
P5	Longword signed	Longword integer (signed)	Read Only	Value	Listener address (optional)

The P1 and P2 parameters specify the address of the data buffer and the number of bytes to be written. The number of bytes actually written is returned in the second word of the I/O status buffer.

If the IEX11 unit issuing the write request is controller-in-charge, and P5 contains a valid **listener address**, then the IEX11 unit will address the device specified by P5 as listener before writing the data. This feature is useful for applications where timing is critical, since it saves the overhead of an extra $QIO to address the device as listener. The device that was addressed as listener remains in the listen state after the write request terminates.

If the unit issuing the write request is controller-in-charge and not addressed as talker when the request is issued, the driver will automatically address it as talker through the **ton** auxiliary command before the data is written. The driver will also send a MTA IEC/IEEE command over the bus to deaddress any other device that is addressed as talker. The driver deaddresses the unit after the transfer has completed.

If the IEX11 unit is in CACS when the write request is issued, the driver will automatically set the unit to CSBS so that the data transfer can occur. It will reset the unit to CACS after the data transfer, unless the IO$M_LEAVIS modifier was specified.

If the unit is not controller-in-charge and not addressed as talker when the request is issued, the driver will wait for the unit to be addressed as talker before initiating the data transfer.

The write request effectively blocks the IEX11 unit. When the unit is addressed as talker after a write request has been issued, the driver does not generate an *addressed as talker* event. However, the driver will automatically issue the **dacr** since the user's process would not be able to issue it.

Note: The automatic features of the IO$_WRITExBLK function (detecting a listener address in P5, issuing the *ton* auxiliary command, and placing the unit in CSBS) minimize the number of required QIOs. However, these features should NOT BE USED in applications where several write requests are required to write a single buffer of data to a device. The proper procedure for such applications is to address the IEX11 unit as talker and the receiving device as listener then put the controller in CSBS before issuing the write requests.

Data can be written through either a direct or buffered data path. The IO$_SETMODE function can be used to select the method best suited for the application. The selection of a data path should be considered fine tuning since it only affects the speed of the transfer. Direct data path is the default.

Modifier	Meaning
IO$M_EOI	Used to assert the EOI line during the transmission of the last byte. The EOI line is used to signal the end of the transfer to the listener.
IO$M_ENA_SRQ	Used to allow service requests to terminate the data transfer. Logic in the IEX11 hardware automatically terminates DMA transfers when an interrupt occurs. The IEX11 driver, therefore, clears the interrupt mask before starting DMA transfers, thus preventing interrupts from terminating the data transfer. This modifier causes the driver to set the service request bit in the interrupt mask register prior to initiating the DMA transfer, thereby allowing service requests to interrupt the data transfer. If a service request interrupts a data transfer, the write request will be returned with the SS$_DEVREQERR status code.
IO$M_CONTROL	Used to leave the controller-in-charge in CACS when the transfer completes. This modifier may only be used for a unit which is CIC.
IO$M_STANDBY	Used to leave the controller-in-charge in CSBS when the transfer completes. This modifier may only be used for a unit which is CIC.
IO$M_LEAVIS	Used to leave the controller-in-charge in the state which was required to process the request. This modifier may only be used for a unit which is CIC.

3.3.18.1 IO$_WRITE{L,P,V}BLK I/O Status Block

Figure 3–11 shows the contents of the I/O status block returned by the driver. The second word of the I/O status block will contain the number of bytes written.

Figure 3–11 IO$_WRITE{L,P,V}BLK I/O Status Block

<table>
<tr><td colspan="4">number of bytes written</td><td>return status</td></tr>
<tr><td>reserved</td><td>addressed state</td><td>reserved</td><td>contr state</td><td>IEC/IEEE status register</td></tr>
<tr><td>31</td><td>27</td><td>23</td><td>19</td><td>15 0</td></tr>
</table>

Refer to Section 3.2 for a description of the contents of the IOSB which are not specific to this request.

3.3.18.2 IO$_WRITE{L,P,V}BLK Status Codes

The I/O status codes are returned in the low word of the first longword of the IOSB or by the SYS$QIO call. They are shown below.

Status Codes	Returned by	Meaning
SS$_NORMAL	IOSB,QIO	Successful completion.
SS$_DATACHECK	IOSB	Request was terminated as a result of an IFC or ERR interrupt.
SS$_DEVACTIVE	QIO	Request issued to a unit which is in diagnostic mode.
SS$_DEVREQERR	QIO	Request issued to a unit which is not initialized.
SS$_DEVREQERR	IOSB	Request was terminated as a result of a service request (IO$M_ENA_SRQ modifier required)
SS$_ILLIOFUNC	IOSB	The request was issued to a unit that is not controller-in-charge.
SS$_IVADDR	IOSB	The buffer specified by P5 contains an invalid listener address.
SS$_TIMEOUT	IOSB	The request failed to complete within the specified time period.

3.3.19 Specifying the P1 parameter in MACRO-32

Care must be taken when using the $QIO_S and $QIOW_S system service routines in MACRO-32 programs. These VMS System services expect the P1 parameter to specify a buffer address. Several requests to the IEX-VMS-Driver, however, specify a single parameter value for P1. These requests are:

IO$_AUXILIARY
IO$_AUXILIARY$
IO$_COMMAND
IO$_COMMAND$
IO$_INITIALIZE
IO$_INITIALIZE$
IO$_LOADPARPOLL
IO$_LOADPARPOLL$
IO$_PASCONTROL
IO$_PASCONTROL$
IO$_SERVICE
IO$_SERVICE$
IO$_SETEVENT
IO$_SETEVENT$
IO$_SETMODE

Higher level languages such as FORTRAN and BASIC take care of this problem by passing the parameters by reference. MACRO-32 has not built in mechanism for passing parameters by reference. However, the following procedures demonstrate how MACRO-32 programs can specify single parameters in P1.

- P1 can be expressed as a literal in the following manner:

```
MASK = ^O67
.
.
.
P1 = MASK,-
P2 = #0
```

- If the value for the P1 parameter is specified in a location, the following coding technique can be used:

```
MASK: .LONG ^O67
.
.
.
P1 = @MASK,-
P2 = 0
```

3.3.20 Using Attention ASTs

There are two methods of detecting events on the IEC/IEEE bus, IO$_REC_EVENT and attention ASTs. The two methods are independent. The IO$_REC_EVENT method has the advantage that it does not require synchronization between an AST routine and your process. It is easier to use since you do not have to code an AST routine. However, event detection is sometimes disabled when the IO$_REC_EVENT method is chosen. For example, the IEX-VMS-Driver blocks event detection during data transfers.

Attention ASTs have the advantage that the IEX11 unit is enabled for event detection in those situations in which IO$_REC_EVENT would be blocked. An attention AST can be outstanding and the IEX11 unit can still be used to process other requests.

You should choose the method best suited for your application and expertise. Attention ASTs take precedence over the IO$_REC_EVENT function.

NOTE: The two methods of event detection MUST NOT BE MIXED.

The attention AST mode of event recognition is selected through the IO$_SETMODE function with the IO$M_ATTNAST function modifier. The P1 parameter of the IO$_SETMODE function specifies the address of the routine which will process the event ASTs. The driver will queue an attention AST to your process when one of the events enabled by a previous IO$_SETEVENT function occurs.

When the event occurs, an AST parameter longword containing event related information is made available to the AST routine. Figure 3–12 shows the contents of the AST parameter longword returned by the driver.

Figure 3–12 Attention AST Parameter

IEX11 controller name	IEX11 unit number	event related information	event number
31 24	16	8	0

The low byte of the AST parameter contains the number of the event that occurred.

The second byte of the AST parameter contains event related information

Refer to Section 3.3.12.1 for a complete description of the event numbers and event related information which may be returned by the driver.

The third byte of the AST parameter contains the IEX11 unit number of the IEX11 controller. This value will be either 0 or 1. This parameter allows one AST routine to process event ASTs from two IEX11 units.

The fourth byte of the AST parameter contains the ASCII value representing the IEX11 controller name (A, B, C, or D, corresponding to IXA, IXB, IXC, and IXD respectively).

Each IO$_SETMODE request with an IO$M_ATTNAST modifier generates only one attention AST. Once an attention AST has been executed, it must be reenabled through another IO$_SETMODE request with the IO$M_ATTNAST modifier before another attention AST can occur.

Your process can queue multiple AST requests. The ASTs will be returned to your process as separate ASTs as events occur.

Issuing the IO$_SETMODE request with the IO$M_ATTNAST function modifier and the P1 parameter equal to zero will cancel all outstanding attention AST requests. Canceling I/O on a channel does not flush outstanding ASTs. All outstanding ASTs are flushed when a process exits. A $DEASSIGN directive for a channel will also flush all outstanding ASTs for that channel.

The following is an example of a simple MACRO-32 AST routine:

```
AST:    .WORD   0               ;AST mask
ASTPRM: .LONG   0               ;AST Parameter

        MOVL    4(AP),ASTPRM    ;Save AST parameter
        RET                     ;Return from AST
```

The IEX exerciser program allows you to select whether events are recognized through the IO$_REC_EVENT request or through attention ASTs. Refer to subroutine SUBAST in IEX.FOR for an example of an AST routine implemented in FORTRAN.

4 SIMPLIFIED USER INTERFACE

4.1 OVERVIEW

The Simplified User Interface (SUI) is a collection of subroutines which provide easy access to the IEX-VMS-Driver. The subroutines can be called from any higher level language that supports the VMS calling standard. The SUI is particularly useful to you if you are familiar with a language's subroutine calling mechanism, but are not familiar with the $QIO system service routines. In general, each subroutine in the SUI package performs one logical IEEE bus function. SUI calls are translated into one or more $QIO calls to the IEX-VMS-Driver.

Note: The use of the SUI subroutine calls does not preclude the use of the $QIO interface in a program. However, it is not recommended that $QIO requests be mixed with SUI calls. If you decide to include $QIO requests in an SUI program, reference the SUI source (IEX$SUI.MAR) to prevent conflicts.

The SUI is made up of the following subroutines:

- **IECMD** - Send IEC/IEEE commands
- **IEPOLL** - Set up a serial poll
- **IERECV** - Receive data from a talker
- **IERQSV** - Issue a service request
- **IESEND** - Send data to one or more listeners
- **IESTRT** - Initialize an IEX11 unit
- **IEWFE** - Wait for an IEX11 IEEE event

It is the responsibility of the application to check the status information returned in the ISTAT buffer after each call.

All ISTAT values specified in this chapter are in addition to those discussed in Chapter 3.

IECMD - Send IEC/IEEE Commands

This subroutine allows a unit that is controller-in-charge to send IEC/IEEE commands to other devices on the bus. In addition, this subroutine can be used to perform certain auxiliary command functions and to provide information about the current state of the controller.

FORMAT

IECMD*(IMPUR ,LTNBUF ,[LTNCNT] ,[CMD] ,ISTAT)*

RETURNS

Information on the status of the call is returned in the ISTAT argument.

ARGUMENTS

IMPUR

VMS Usage: **longword_unsigned**
type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Address of the data area for the unit. Specified in IESTRT.

LTNBUF

VMS Usage: **longword_unsigned**
type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Address of byte array containing the addresses (primary or primary and secondary) of the devices to which the CMD command is directed **OR** address of the byte array containing the IEX commands to be output.

LTNCNT

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Length of the LTNBUF array in bytes (optional)

CMD

VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by reference**

Three character ASCII string indicating the command to be performed (optional)

ISTAT

VMS Usage: **longword_unsigned**
type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Address of a four word integer buffer into which status information concerning the call is returned.

The first word of the ISTAT array indicates whether the call was successful. The following values have specific meanings for the SUI.

Value	Meaning
1.	Normal successful return
9.	Call terminated by service request (possible only if IEXFL parameter in IESTRT is specified)
20.	Bad parameter count
244.	Request was issued to a unit that was not controller-in-charge

Any other error code in this word as generated by the operating system. The LIB$SIGNAL run-time library routine can be used to evaluate messages generated by the operating system.

The second word of ISTAT contains the subroutine specific information. If the CMD parameter is AS? or CS?, the requested information is returned in this word. For AS?, the following values may be returned:

0 - the unit is not addressed
1 - the unit is addressed as listener
2 - the unit is addressed as talker

For CS?, the following values may be returned:

1 - the unit is in the Controller Idle State
2 - the unit is in the Controller Active State
3 - the unit is in the Controller Standby State

DESCRIPTION

The IECMD subroutine can be used to perform a single command function or to output a buffer of IEC/IEEE commands.

The usual way in which IECMD is used is to specify the command to be performed in the CMD argument. If the command requires the device(s) to be addressed as listener(s), then the listener addresses are specified in LTNBUF. The following ASCII strings can be specified in the CMD argument:

Command	Meaning
IFC	Send interface clear
REN	Send remote enable[2]
LCL	Clear remote enable
GTL	Send go to local[1]
LLO	Send local lockout
SDC	Send selected device clear[1]
DCL	Send device clear
GET	Send group execute trigger[1]
UNL	Unlisten all devices
UNT	Untalk all devices
GTS	Go to standby
GCT	Go to Controller Active State
RLS	Release DAC hold-off
AS?	Return addressed state
CS?	Return controller state

[1]Subroutine will send an UNL command to unlisten all devices, address the devices specified in the LTNBUF array as listeners, and then send the command. The listeners remains addressed as listeners unless a specific UNL command is issued.

[2]Subroutine will issue **sre** auxiliary command, send the buffer of listener addresses, and then issue a UNL command.

IFC, REN, LCL, and RLS are auxiliary commands rather than IEC/IEEE commands. They are processed by the IO$_AUXILIARY$ request. GTS and GCT cause the IO$_GO_TO_CSBS$ and IO$_GO_TO_CACS$ requests, respectively, to be executed. AS? and CS? are processed by the IO$_SENSEMODE request and return the requested information in the second word of the ISTAT argument. The commands discussed in this paragraph are the only non-IEC/IEEE commands that can be processed by the SUI.

The commands marked by a 1 or 2 require that the devices be addressed as listener in order to be affected by the command. For these commands LTNBUF is required. All values in the buffer less than 40_8 will automatically be converted to listener addresses before being transmitted. The array may contain both primary addresses and secondary addresses. LTNCNT specifies the length of the LTNBUF array in bytes. A primary-secondary address pair counts as two bytes.

The IECMD subroutine can also be used to output a buffer of IEC/IEEE commands. To do this, omit the CMD argument and place the values of the IEC/IEEE commands to be executed in the LTNBUF array. By following this procedure, you can output any combination of IEC/IEEE commands, even those not covered by the CMD argument. Note that only IEC/IEEE commands can be placed in the buffer. Refer to Appendix C for a listing of IEC/IEEE commands.

IEPOLL - Set Up a Serial Poll

This subroutine is used by the controller-in-charge to set up a serial poll. IEPOLL specifies the addresses of the devices which will be addressed in the serial poll and a buffer where the results of the serial poll will be stored. When a service request is detected by the SUI package (usually through the IEWFE subroutine), the serial poll will automatically be performed. IEPOLL is usually called only once in a program.

FORMAT

IEPOLL*(IMPUR ,SPLST ,SPRBUF ,SPNUM ,SPSTAT ,ISTAT)*

RETURNS

Information on the status of the call is returned in the ISTAT argument.

ARGUMENTS

IMPUR

VMS Usage: **longword_unsigned**
type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Address of the data area for the unit. Specified in IESTRT.

SPLST

VMS Usage: **longword_unsigned**
type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Address of a byte array containing the primary (and optional secondary) addresses of devices to be addressed in the serial poll.

When the SUI detects that a device has requested service, the SUI package automatically performs a serial poll to determine which device on the IEC/IEEE bus requested service. The SUI will automatically convert primary addresses to talker addresses.

SPRBUF

VMS Usage: **longword_unsigned**
type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Address of a byte array where the status bytes of the devices which have been polled are stored.

As the serial poll is conducted, the status bytes returned by the devices that are polled are stored in this buffer. The bytes are stored in the same order as the device addresses in SPLST. Your program can determine which device(s) requested service by examining bit six of the bytes in this array. Bit six of the status byte for the device(s) that requested service will be set. The low five bits and bit seven may contain device specific information indicating the reason the device(s) requires service.

SPNUM

VMS Usage:	**longword_signed**
type:	**longword integer (signed)**
access:	**read only**
mechanism:	**by reference**

The length of SPLST in bytes. Primary-secondary address pairs count as two bytes.

SPSTAT

VMS Usage:	**longword_unsigned**
type:	**longword integer (unsigned)**
access:	**read only**
mechanism:	**by reference**

Address of a two word integer array containing serial poll status information.

The two words in this array are initialized to zero when this routine is called. If the first word has a value of 0, service request interrupts are enabled and a serial poll will be conducted the next time a device requests service. If the first word has a value of 1, a serial poll has been conducted and service requests are disabled. You must set SPSTAT(1) = 0 in order for another serial poll to be performed. If the first word has some other value, an error occurred during the serial polling. The second word in the array returns the number of devices that were polled.

ISTAT

VMS Usage: **longword_unsigned**
type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Address of a four word integer buffer into which status information concerning the call is returned.

The first word of the ISTAT array indicates whether the call was successful. The following values have specific meanings for the SUI.

Value	Meaning
1.	Normal successful return
9.	Call terminated by service request (possible only if IEXFL parameter in IESTRT is specified)
20.	Bad parameter count
244.	Request was issued to a unit that was not controller-in-charge
308.	Invalid address has been included in SPLST

Any other error code in this word as generated by the operating system. The LIB$SIGNAL run-time library routine can be used to evaluate messages generated by the operating system.

DESCRIPTION

Once IEPOLL has been called, the SUI will perform a serial poll whenever a device requests service, as long as you set SPSTAT(1) = 0 initially and after each poll is performed. However, the SUI allows the program to either continue executing SUI calls after the serial poll has been set up or to wait for a device to request service. If you decide to continue calling SUI subroutines, you have an additional choice to make. You must decide if you want the serial poll to be conducted as soon as the currently executing SUI call has completed, or if you want to terminate the currently executing call and have the serial poll conducted immediately. The IEXFL parameter in the IESTRT subroutine determines whether or not a service request will terminate the currently executing SUI call.

The usual way for recognizing service requests is to take the default value for IEXFL (allowing currently executing SUI calls to complete before the serial poll is conducted) and to call the IEWFE subroutine to wait for a device to request service. As soon as a device requests service, the IEWFE subroutine will complete, and the serial poll will be conducted. Your program can then inspect SPLST to determine which device(s) requested service and take appropriate action.

The second way that serial polling can be set up also takes the default value for IEXFL. However, rather than calling IEWFE to wait for service requests, the program continues to execute. If a device requests service, the currently executing SUI subroutine completes, and the serial poll is immediately executed. Your program must periodically check SPSTAT(1) to determine if a serial poll has occurred. If SPSTAT(1) = 1, then a serial poll has been conducted.

The final way to set up serial polling is to not accept the default value of IEXFL. This causes SUI requests to be immediately aborted with a status of nine and the serial poll to be conducted. Your program can determine if a serial poll has occurred by checking ISTAT(1) of SUI calls or SPSTAT(1). If ISTAT(1) = 9. or SPSTAT(1) = 1, then a serial poll has occurred. You must determine the appropriate action to take for an aborted SUI call.

For most applications, defaulting IEXFL and using IEWFE to wait for service requests is the appropriate way to handle serial polling. However, one of the other procedures may be more appropriate under certain conditions. Which procedure is best depends upon such things as the number of devices on the bus, the frequency that devices request service, the speed of the devices, and the size of the data transfers.

For all of the above procedures, your program must set SPSTAT(1) = 0 after each serial poll has been conducted. Otherwise, no additional serial polls will be performed.

IERECV - Receive Data From A Talker

This subroutine is used to read data from another device on the IEC/IEEE bus and store it in an array. The subroutine can also address the sending device as a talker by specifying the optional TALKER parameter. In this case, the subroutine will untalk and unlisten all devices, address itself as a listener, address the selected device as talker, and read the data.

FORMAT

IERECV*(IMPUR ,BUFF ,BYTCNT ,[MATCHR] ,[MATCNT] ,[TALKER] ,ISTAT)*

RETURNS

Information on the status of the call is returned in the ISTAT argument.

ARGUMENTS

IMPUR

VMS Usage: **longword_unsigned**
type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Address of the data area for the unit. Specified in IESTRT.

BUFF

VMS Usage: **longword_unsigned**
type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Address of a byte array where the data which is read will be stored.

The data array may contain either ASCII characters or numerical ASCII equivalents in binary, octal, decimal, or hexidecimal.

BYTCNT

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

The length of BUFF in bytes. This is the maximum number of bytes that will be read into BUFF. Range: 1...65535

MATCHR

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Address of a one word buffer in which the low order byte contains the match character (optional).

This argument specifies the character on which the read will terminate.

MATCNT

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

The match character count (optional).

This argument specifies the consecutive number of times the match character must be read to terminate the read request.

TALKER

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Address of a one word buffer containing the talker address (optional).

If the device reading the data is controller-in-charge, the low order byte of this argument may be used to specify the address of the device which will be sending the data. The second byte of this argument may be used to specify the extended talker address of the device which will be sending the data.

ISTAT

VMS Usage: **longword_unsigned**
type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Address of a four word integer buffer into which status information concerning the call is returned.

The first word of the ISTAT array indicates whether the call was successful. The following values have specific meanings for the SUI.

Value	Meaning
1.	Normal successful return
9.	Call terminated by service request (possible only if IEXFL parameter in IESTRT is specified)
20.	Bad parameter count
92.	Transfer terminated as a result of an IFC (interface clear) or ERR (error) interrupt

Any other error code in this word as generated by the operating system. The LIB$SIGNAL run-time library routine can be used to evaluate messages generated by the operating system.

The second word of the ISTAT array, ISTAT(2), contains the number of bytes that were read.

DESCRIPTION

The three ways in which a read can be terminated are:

- When EOI is asserted
- When MATCNT consecutive occurrences of the value specified in MATCHR have been read
- When BYTCNT number of bytes have been read

IERQSV - Issue A Service Request

This subroutine allows an IEX11 unit that is not controller-in-charge to issue a service request.

FORMAT

IERQSV*(IMPUR ,IRSVST ,ISTAT)*

RETURNS

Information on the status of the call is returned in the ISTAT argument.

ARGUMENTS

IMPUR

VMS Usage: **longword_unsigned**
type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Address of the data area for the unit. Specified in IESTRT.

IRSVST

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Address of one word buffer in which the low byte contains the serial poll status byte. This byte will be loaded into the serial poll register.

ISTAT

VMS Usage: **longword_unsigned**
type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Address of a four word integer buffer into which status information concerning the call is returned.

The first word of the ISTAT array indicates whether the call was successful. The following values have specific meanings for the SUI.

Value	Meaning
1.	Normal successful return
9.	Call terminated by service request (possible only if IEXFL parameter in IESTRT is specified)
20.	Bad parameter count

Any other error code in this word as generated by the operating system. The LIB$SIGNAL run-time library routine can be used to evaluate messages generated by the operating system.

DESCRIPTION

If bit six, (**rsv1**) in the low order byte of IRSVST is set, then the IEX11 unit will assert the service request line when this subroutine is called. If the call is made with bit six clear, the serial poll status byte will be loaded into the serial poll register, but no service request will be made. This byte is read by the controller-in-charge when a serial poll is performed.

IESEND - Send Data to One or More Listeners

This subroutine is used to send data to other devices on the IEC/IEEE bus. If the IEX11 unit that will be sending the data is controller-in-charge, the subroutine can also be used to address the device(s) that will receive the data as listener. In this case, before sending the data, the controller-in-charge will untalk and unlisten all devices, address the selected device(s) as listener, and address itself as talker.

FORMAT

IESEND*(IMPUR ,BUFF ,BYTCNT ,[LTNBUF] ,[LTNCNT] ,ISTAT)*

RETURNS

Information on the status of the call is returned in the ISTAT argument.

ARGUMENTS

IMPUR

VMS Usage: **longword_unsigned**
type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Address of the data area for the unit. Specified in IESTRT.

BUFF

VMS Usage: **longword_unsigned**
type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Address of a byte array containing the data to be written.

The values in BUFF may be ASCII characters or equivalents specified in binary, octal, decimal, or hexadecimal.

BYTCNT

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

The length of BUFF in bytes (optional).

This is the maximum number of bytes that will be written. This value should be equal to or less than the number of bytes in the BUFF array. If this argument is omitted, the SUI assumes that BUFF contains an ASCII string terminated by a zero byte and computes the byte count. Range: 1...65535

LTNBUF

VMS Usage: **longword_unsigned**
type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Address of a byte array containing the listener addresses of the devices to receive the data (optional).

If this call is being made to an IEX11 unit that is controller-in-charge, the LTNBUF array can contain the listener address of the device(s) to receive the data. The array can contain either primary addresses or primary/secondary address pairs. The subroutine will automatically convert primary addresses to listener addresses.

LTNCNT

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

The length of LTNBUF in bytes (optional, required if LTNBUF is specified).

Primary-secondary address pairs count as two bytes.

ISTAT

VMS Usage: **longword_unsigned**
type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Address of a four word integer buffer into which status information concerning the call is returned.

The first word of the ISTAT array indicates whether the call was successful. The following values have specific meanings for the SUI.

Value	Meaning
1.	Normal successful return
9.	Call terminated by service request (possible only if IEXFL parameter in IESTRT is specified)
20.	Bad parameter count
92.	Transfer terminated as a result of an IFC (interface clear) or ERR (error) interrupt

Any other error code in this word as generated by the operating system. The LIB$SIGNAL run-time library routine can be used to evaluate messages generated by the operating system.

The second word of the ISTAT array, ISTAT(2), contains the number of bytes that were written.

DESCRIPTION

The SUI sends the data using the IO$_WRITELBLK function with the IO$M_EOI modifier. As a result, the EOI line is always asserted when the last byte of data is written.

IESTRT - Initialize an IEX11 Unit

This subroutine is used to initialize an IEX11 unit. IESTRT must always be the first SUI subroutine called for the unit. The subroutine can also be called to detach from a unit and to clean up before your process terminates.

FORMAT

IESTRT*(IMPUR ,CHAN ,[IEFL] ,[IEXFL] ,IADDR ,ICTFLG ,ISTAT)*

RETURNS

Information on the status of the call is returned in the ISTAT argument.

ARGUMENTS

IMPUR

VMS Usage: **longword_unsigned**
type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Address of an 80-byte array for storing unit dependent information. This argument sets aside an 80-byte area in your process for storing unit dependent information. Each IEX11 unit which an SUI program accesses must have a unique IMPUR area. The other SUI subroutines use the information stored in the impure area to identify the unit to which the request is directed. This area must not be modified by your program.

CHAN

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

This parameter specifies the channel number of the IEX11 unit being initialized. Before calling IESTRT, the program must use the $ASSIGN system service to allow VMS to assign a channel to the unit. IESTRT stores this channel number in the IMPUR area, and subsequent calls to the SUI obtain the channel number of the IEX11 unit to which the call is directed from its IMPUR area.

IEFL

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Event flag for the $QIO system service (optional)

The SUI subroutines use the $QIOW_S system service to issue requests to the driver. This system service requires that an event flag be specified so that control returns only after the $QIO request has completed. It is recommended that you use the LIB$GET_EF Run-Time Library routine to allocate a unique event flag for this subroutine. Before exiting, your program should issue a LIB$FREE_EF call to return the event flag to the system. If your program does not use event flags for other purposes, this parameter may be omitted or set to zero. If the parameter is omitted or zero, the SUI will issue its own call to LIB$GET_EF.

IEXFL

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

Service request event flag (optional).

This parameter is only applicable for a unit that is controller-in-charge. Specifying this event flag allows service requests to interrupt other calls to the SUI. If specified, this event flag should be allocated from the system in the same manner as IEFL. It MUST NOT BE the same event flag as specified in the IEFL parameter. The usual way for recognizing service requests is to call IEWFE and wait for a device to request service. If service requests are to be identified in this way, this parameter should be omitted. See the description of IEPOLL for more information about how this parameter affects serial polls.

IADDR

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

IEC/IEEE bus address for the IEX11 unit.

The IEC/IEEE bus address for the IEX11 unit is software selectable. Valid bus addresses are 0 to 36_8. Each device on the IEC/IEEE bus must have a unique address in this range. ICTFLG must be equal to or less than one to set this address.

ICTFLG

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

This parameter specifies the controller state that the unit should enter. It can assume the following values:

Value	Meaning
ICTFLG = 1	IEX11 unit is initialized to system controller, CIC, and CACS
ICTFLG < 1	IEX11 unit is initialized to non CIC
ICTFLG > 1	Issues **dacr** and disables event recognition

ISTAT

VMS Usage: **longword_unsigned**
type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Address of a four word integer buffer into which status information concerning the call is returned.

The first word of the ISTAT array indicates whether the call was successful. The following values have specific meanings for the SUI.

Value	Meaning
1.	Normal successful return
20.	Bad parameter count
244.	The device assigned to CHAN is not an IEX11 unit

Any other error code in this word as generated by the operating system. The LIB$SIGNAL run-time library routine can be used to evaluate messages generated by the operating system.

IEWFE - Wait for an IEX11 IEEE Event

This subroutine is used to wait for an event on the IEC/IEEE bus. A unit that is controller-in-charge will wait for the occurrence of either of the following events:

- Service request
- Interface clear

A unit that is not controller-in-charge will wait for the occurrence of one of the following events:

- Addressed as listener
- Addressed as talker
- Device clear
- Device trigger
- Interface clear

Your program cannot select which event to wait for. If an IEWFE call is issued to an IEX11 unit that is controller-in-charge, your program will hang until either a service request or an interface clear occurs. The subroutine returns information specifying the event that occurred in one of the parameters of the call.

FORMAT

IEWFE*(IMPUR ,IEVENT ,ISTAT)*

RETURNS

Information on the status of the call is returned in the ISTAT argument.

ARGUMENTS

IMPUR

VMS Usage: **longword_unsigned**
type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Address of the data area for the unit. Specified in IESTRT.

IEVENT

VMS Usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

This parameter contains an event code indicating the event that occurred. The event codes returned are shown below. They have the same values as the event codes returned by the IO$_REC_EVENT function.

Value	Event
1.	Service request
2.	Addressed as listener
3.	Addressed as talker
7.	Device clear
8.	Group execute trigger
13.	Interface clear

ISTAT

VMS Usage: **longword_unsigned**
type: **longword integer (unsigned)**
access: **read only**
mechanism: **by reference**

Address of a four word integer buffer into which status information concerning the call is returned.

The first word of the ISTAT array indicates whether the call was successful. The following values have specific meanings for the SUI.

Value	Meaning
1.	Normal successful return
9.	Call terminated by service request (possible only if IEXFL parameter in IESTRT is specified)
20.	Bad parameter count

Any other error code in this word as generated by the operating system. The LIB$SIGNAL run-time library routine can be used to evaluate messages generated by the operating system.

DESCRIPTION

If the controller-in-charge issues an SDC or GET to an IEX11 unit that is not controller-in-charge, the IEX11 unit will first be addressed as a listener. This will generate an *addressed as listener* event in the driver. At this point, it is unclear whether the event which will occur is *device clear*, *device trigger*, or simply *addressed as listener*. The SUI resolves this dilemma in the following manner. When an *addressed as listener* event is detected, the SUI waits until either a command is received, or the attention line goes false. If a command is received, the event (*device clear* or *device trigger*) corresponding to the command is returned. If the ATN line is deasserted, the unit has entered the Listener Active State, and an *addressed as listener* event is returned.

5 IEX EXERCISER PROGRAM

The IEX Exerciser is a program which allows you to interactively access the IEX-VMS-Driver. The exerciser can also be used to execute a series of commands contained in an indirect command file. The development time to write IEX11 programs can often be significantly reduced by using the exerciser to confirm that an IEEE-488 device responds in the manner you think it does. Translating an exerciser command file into an SUI or $QIO program is usually a simple and straightforward operation.

5.1 RUNNING THE IEX EXERCISER PROGRAM

The exerciser program is located in the [.IEX] subdirectory of SYS$EXAMPLES. To run the program from your default directory, enter the following command:

```
$RUN SYS$SYSROOT:[SYSHLP.EXAMPLES.IEX]IEX
```

At the **'IEX$>'** prompt, either begin entering exerciser commands interactively, or enter the name of the exerciser command file to be executed.

The indirect command file IEX$DEMO.COM is also located in the [.IEX] subdirectory of SYS$EXAMPLES. This command file demonstrates most of the I/O requests supported by the IEX-VMS-Driver. Successful execution of IEX$DEMO.COM provides a good test that both the IEX11 hardware and software are correctly installed and operating normally.

In order for the IEX$DEMO.COM command file to be executed properly, the two units: IXA0: and IXA1: must be connected to each other with a flat test cable (BC08S-01), or they must be connected to each other on the same IEC/IEEE bus. IEX$DEMO.COM assigns the two units IEC/IEEE bus addresses 0 and 1 respectively. You must be sure that there are no other devices on the IEC/IEEE bus with these addresses.

By default, the exerciser program attempts to assign channels to the first four IEX11 controllers (8 IEX11 units) on the system (IXA0:, IXA1:, IXB0:, IXB1:, IXC0:, IXC1:, IXD0, and IXD1:). However, IEX$DEMO.COM only attempts to use the first two units (IXA0: and IXA1:). If you wish to utilize units other than the IXA0: and IXA1:, you must modify IEX$DEMO.COM to address the desired units.

At the **'IEX$>'** prompt, enter the following command to execute the IEX$DEMO.COM command file:

```
IEX$>@SYS$SYSROOT:[SYSHLP.EXAMPLES.IEX]IEX$DEMO
```

The IEX$DEMO.COM command file produces a lengthy printout with many comments. Any errors returned by the IEX-VMS-Driver will be displayed. Appendix A contains a complete listing of the output produced by executing IEX$DEMO.COM.

5.2 IEX EXERCISER COMMANDS

Commands to the IEX exerciser program are very similar to the I/O requests processed by the driver. In general, the command string consists of an abbreviation for the I/O function, the unit to which the request is to be made, and a list of parameters for the $QIO.

Consider the following exerciser command:

```
IEX$>SPO 0 101
```

The string **SPO** indicates that a serial poll request is to be made to the driver. The **0** specifies that the request will be made to IXA0:, and the **101** indicates that the device to be polled has a my talk address, MTA, of 101_8.

Some exerciser commands have several different abbreviations. These alternate abbreviations are included to provide compatibility with the RSX version of the IEX exerciser. Unless otherwise noted, the different command abbreviations produce identical results. Modifiers for functions may be ORed with the IEX command by using an !. For example, with the command:

```
IEX$>SPO!IO$M_ALLDEVICES 0 101
```

The driver performs a serial poll from IXA0: regardless of the state of the service request line.

The argument following the command string specifies the controller to which the request will be issued. The IEX exerciser supports up to eight IEX11 units: IXA0:, IXA1:, IXB0:, IXB1:, IXC0:, IXC1:, IXD0, and IXD1:. A 0 indicates that the request will be issued to unit IXA0:, a 1 indicates that it will be issued to unit IXA1:, and so on. A 7 indicates that the request will be issued to unit IXD1:

By default, the exerciser assumes that all numbers are octal. Numbers can be expressed in decimal radix by appending a decimal point to them, or hexadecimal radix by appending the character X to them. Numbers which contain characters A through F are automatically converted to hexadecimal.

When the exerciser is run in loopback mode, several commands require a corresponding command to be issued to the other unit before they can complete. For example, a service request to one unit requires a serial poll command to be issued to the other unit before it will complete. In order to be able to issue the serial poll command, the exerciser must return to the **IEX$>** prompt. The exerciser, therefore, issues a $QIO request rather than a $QIOW request for service requests. This allows the exerciser program to continue executing other commands rather than waiting for the service request to complete. The WFQ command (Wait for Request) may be issued after the service request to stop execution until a serial poll has been completed.

The descriptions of each exerciser command indicate whether or not the exerciser waits for that command to complete before continuing. For the sake of clarity, unit 0 is specified in all of the examples in the following discussions. All arguments must be separated by one space. Exerciser commands can be entered in either upper or lower case.

Table 5–1 lists the exerciser commands which are described in detail in the following sections.

Table 5–1 IEX Exerciser Commands Summary

Exerciser Commands	IEX[1] unit number	Parameters[2]	I/O Request Generated
AST	0	ENABLE or DISABLE	IO$_SETMODE!IO$M_ATTNAST
AUX	0	cmd	IO$_AUXILIARY$
CAN,KILL	0	n/a	$CANCEL (system service)
CMD	0	arg1 [arg2 ...]	IO$_COMMAND$
DISABLE	n/a	CHK	n/a
ENABLE	n/a	CHK	n/a
EXIT	n/a	n/a	n/a
GCT	0	n/a	IO$_GO_TO_CACS$
GTS	0	n/a	IO$_GO_TO_CSBS$
INI,STS	0	mask devnr	IO$_INITIALIZE$
LPP	0	sttbyt	IO$_LOADPARPOLL$
PCT	0	tlkadr	IO$_PASSCONTROL$
PPC	0	lisadr1 cnfbyt1 [...]	IO$_PARPOLLCON$
PPO	0	n/a	IO$_PAR_POLL$
PRINT	n/a	[format[2]]	n/a
RDA,RLB,RLM	0	bytcnt [matchr] [matcnt]	IO$_READLBLK
RES)	0	n/a	IO$_REC_EVENT$!IO$M_NOWAIT
REV	0	n/a	IO$_REC_EVENT$
RSV	0	status extadr	IO$_SERVICE$
SEM	0	mask	IO$_SETEVENT$
SET DATA	n/a	format[2] offset data1 [data2...]	n/a
SMD!modifier	0	parameter	IO$_SETMODE!modifier
SNS,SNI	0	n/a	IO$_SENSEMODE
SPO	0	tlkadr1 [extadr1] ...	IO$_SER_POLL$
STO,TMO	0	seconds	IO$_SETMODE!IO$M_TIMOUT
VERIFY	n/a	[format[2]] [length]	n/a
WDA,WLB	0	bytcnt or string	IO$_WRITELBLK
WET,WLE	0	bytcnt or string	IO$_WRITELBLK!IO$M_EOI
WFE	0	n/a	n/a
WFQ	0	n/a	n/a

[1]All IEX unit numbers specified above are 0 (IXA0:) for clarity. Applicable IEX unit numbers should be substituted.

[2]Buffer format specification B, C, D, O, W, X

5.2.1 AST - Enable/Disable Attention ASTs

This command is used to enable or disable the reporting of events by the unsolicited AST mechanism. Refer to Chapter 3 for information on the use of attention ASTs to report IEC/IEEE bus events.

Format	Request Generated
AST 0 ENABLE	IO$_SETMODE!IO$M_ATTNAST
AST 0 DISABLE	"

The exerciser waits for this request to complete before continuing.

5.2.2 AUX - Output an Auxiliary Command

This command is used to transmit an auxiliary command. The **cmd** parameter contains an octal number indicating the auxiliary command to be executed. Refer to Table 3–5 for a list of valid auxiliary commands.

Format	Request Generated
AUX 0 **cmd**	IO$_AUXILIARY$

Note that some auxiliary commands require bit seven to be set to activate the command. These commands can be deactivated by issuing the command with bit seven cleared, as shown in the following example:

```
AUX 0 220    ;Sets remote enable
```

The exerciser waits for this request to complete before continuing.

5.2.3 CAN,KIL - Cancel I/O on Channel

This command will cancel all uncompleted requests which have been issued to a unit. Only requests which the exerciser does not wait to complete can be cancelled.

Format	Request Generated
CAN 0	$CANCEL system service
KIL 0	"

The exerciser waits for this request to complete before continuing.

5.2.4 CMD - Output IEC/IEEE Bus Commands

This command is used to transfer a buffer of IEC/IEEE commands. The unit issuing the request must be controller-in-charge.

Format	Request Generated
CMD 0 **arg1 [arg2...]**	IO$_COMMANDS$

Arguments **arg1**, **arg2**, ... are octal numbers representing the IEC/IEEE commands. For example:

```
CMD 0 77 137       ;Issues unlisten and untalk commands
```

The exerciser does not wait for this request to complete before continuing.

5.2.5 DISABLE CHK - Disable Data Checking

This command is the opposite of the ENABLE CHK command. It disables the automatic checking of data during transfers. Data checking tests whether the data input by one IEX11 unit is identical to the data output by another IEX11 unit. Therefore, data checking should be used only when data is being transferred between two IEX11's. Data checking should be disabled if the user is using the exerciser to send or receive data from a non-IEX11 device.

Format	Request Generated
DISABLE CHK	None

5.2.6 ENABLE CHK - Enable Data Checking

This command enables the automatic checking of data during transfers. It is used when testing two IEX11 units in closed loop mode. When a read request for one IEX11 completes, the data received is compared with the data transferred by the other IEX11 unit. An error message is generated if differences are detected.

Format	Request Generated
ENABLE CHK	None

5.2.7 EXIT - Exit

This command causes a clean exit from the IEX exerciser program.

Format	Request Generated
EXIT	None

5.2.8 GCT - Get Control

This command may only be issued by the controller-in-charge. It is used to set the controller to Controller Active State (CACS). If the controller is already in CACS, then the success code is returned and no operation is performed.

Format	Request Generated
GCT 0	IO$_GO_TO_CACS$

The exerciser waits for this request to complete before continuing.

5.2.9 GTS - Go To Standby

This command may only be issued by the controller-in-charge. It is used to set the controller to Controller Standby State (CSBS). If the controller is already in CSBS, then the success code is returned and no operation is performed. The most frequent use of this command is to put the controller-in-charge in CSBS so that data can be transferred between other devices on the IEC/IEEE bus.

Format	Request Generated
GTS 0	IO$_GO_TO_CSBS$

The exerciser waits for this request to complete before continuing.

5.2.10 INI,STS - Initialize Unit

These commands are used to initialize a controller and to assign a primary address to it. The **mask** parameter selects initialization options, and the **devnr** parameter selects the primary IEC/IEEE bus address for the IEX11 unit. Valid IEC/IEEE primary addresses range from 0 to 36_8.

Refer to Table 3–7 for a description of the bits in the **mask** parameter.

Format	Request Generated
INI 0 **mask devnr**	IO$_INITIALIZE$
STS 0 **mask devnr**	"

Example:

```
IEX$>INI 0 67 0  ;Init unit 0 to sys controller, CACS w/ bus addr 0
IEX$>INI 1 43 1  ;Init unit 1 to non-CIC w/ bus addr 1
```

The exerciser waits for this request to complete before continuing.

5.2.11 LPP - Load Parallel Poll Register

This command is used to load a status byte (**sttbyt**) into the parallel poll register. When a parallel poll is conducted, this information is transferred to the controller-in-charge.

Format	Request Generated
LPP 0 **sttbyt**	IO$_LOADPARPOLL$

The exerciser waits for this request to complete before continuing.

5.2.12 PCT - Pass Control

This command allows the controller-in-charge to pass control to another device on the IEC/IEEE bus. The command can only be issued by the controller-in-charge.

Format	Request Generated
PCT 0 **tlkadr**	IO$_PASSCONTROL$

The **tlkadr** parameter contains the talker address of the device to which control is to be passed. When this command completes, the unit which issued the command is in CIDS, and the device specified by the **tlkadr** parameter is both controller-in-charge and CACS.

The exerciser does not wait for this request to complete before continuing.

5.2.13 PPC - Parallel Poll Configure

This command allows the controller-in-charge to configure up to eight devices on the IEC/IEEE bus to participate in a parallel poll.

Format	Request Generated
PPC 0 **lisadr1 cnfbyt1 lisadr2 cnfbyt2 ...**	IO$_PARPOLLCON$

The **lisadr1**, **lisadr2**, ... parameters are the listener addresses of the devices which will participate in the parallel poll. The **cnfbyt1**, **cnfbyt2**, ... parameters are the parallel poll configure bytes for the devices. Bits 0-2 of each **cnfbyt** associates a specific bit (0-7) with the device. If bit three of **cnfbyt** is set, then the device is to respond with a *1* to indicate a true condition. If bit three of **cnfbyt** is clear, a *0* response indicates a true condition for that device.

The following example assigns device address 1 to respond true by bit 3 set to a one:

```
IEX$>PPC 0 41 13    ;device 1 responds true when bit 3 set.
```

The exerciser does not wait for this request to complete before continuing.

5.2.14 PPO - Perform a Parallel Poll

This command causes the controller-in-charge to conduct a parallel poll. The parallel poll status byte is displayed when this request completes.

Format	Request Generated
PPO 0	IO$_PAR_POLL$

The exerciser waits for this request to complete before continuing.

5.2.15 PRINT - Print Data in Read Buffer

This command causes the data from the last read request (RDA, RLB, or RLM) to be displayed. The user can select the format in which the data is printed. The following options are available:

B - Print as byte values
W - Print as word values
O - Print as octal values
D - Print as decimal values
X - Print as hexadecimal values
C - Print as ASCII characters

Format	Request Generated
PRINT [B] [W] [O] [D] [X] [C]	None

The default is to print the buffer as octal byte values.

5.2.16 RDA,RLB,RLM - Read Data

RDA, RLB, and RLM allow an IEX11 unit to read a buffer of data.

Format	Request Generated
RDA 0 **bytcnt [matchr matcnt]**	IO$_READLBLK
RLB 0 **bytcnt [matchr matcnt]**	IO$_READLBLK
RLM 0 **bytcnt [matchr matcnt]**	IO$_READLBLK

The **bytcnt** parameter specifies the maximum number of bytes (up to 16384.) that will be read before the request terminates. Fewer than **bytcnt** bytes may be read if the device sending the data asserts EOI.

Match character termination can be enabled by specifying the optional **matchr** and **matcnt** parameters. If these parameters are specified, the read request can also be terminated when the character specified by **matchr** is consecutively received the number of times specified by **matcnt**.

Note: Refer to Section 3.3.11 for a description of the restriction when the IO$M_TCS modifier is used with match character termination enabled.

The exerciser does not wait for this request to complete before continuing.

5.2.17 RES - Read Event Status

This command checks to see if an event enabled by the SEM command has occurred. It is similar to the REV command except that it completes immediately if no event is queued in the driver.

Format	Request Generated
RES 0	IO$_REC_EVENT$!IO$M_NOWAIT

The exerciser waits for this request to complete before continuing.

5.2.18 REV - Recognize Event

This command is used to recognize events. Before an event can be returned by this request, recognition for the event must be enabled through the SEM command. If an enabled event occurs before this command is issued, the event is saved in the driver and returned when the command is issued. The WFE command is used to wait for the occurrence of an event and to print the event related information.

Format	Request Generated
REV 0	IO$_REC_EVENT$

The following exerciser commands would need to be entered to have IXA0: wait for an *addressed as talker* event:

```
IEX$>SEM 0 4   ;Event of addressed as talker
IEX$>REV 0     ;Recognize event
IEX$>WFE 0     ;Wait for event
```

The exerciser does not wait for this request to complete before continuing.

5.2.19 RSV - Request Service

This command allows an IEX11 unit to generate a service request to the controller-in-charge.

Format	Request Generated
RSV 0 **status [extadr]**	IO$_SERVICE$

The byte value specified by the **status** parameter is loaded into the serial poll status register for the unit. When the device is addressed as a talker during a serial poll, this value will be output on the IEC/IEEE bus and read by the controller-in-charge. If bit six (**rsv1**) of **status** is set, the SRQ line will be asserted. If bit six is cleared, the status byte will be loaded into the serial poll register, but no service request will be generated.

If the optional **extadr** parameter is specified, extended addressing will be enabled during the serial poll. The unit will not output the serial poll status byte until both its primary and secondary addresses have been specified.

The exerciser does not wait for this request to complete before continuing.

5.2.20 SEM - Set Event Mask

This command specifies the events for which recognition is to be enabled. Refer to Section 3.3.16 for a complete description of the event mask bits.

Format	Request Generated
SEM 0 **mask**	IO$_SETEVENT$

Recognition of several different types of events may be enabled at any given time.

Example:

```
IEX$>SEM 0 22      ;Enable addressed as listener or extended listener
```

The exerciser waits for this request to complete before continuing.

5.2.21 SET DATA - Set Data in Write Buffer

This command allows the user to fill the internal write buffer with application dependent data. The data can then be output by the WDA, WET, WLB, or WLE commands. The user can select the format in which the data is entered. The following options are available:

B - Data specified as byte values
W - Data specified as word values
O - Values specified in octal format
D - Values specified in decimal format
X - Values specified in hexadecimal format
C - Values specified in ASCII character format

Format	Request Generated
SET DATA [B] [W] [O] [D] [X] [C] **offset data1 ... datan**	None

Octal byte values can be specified in character strings by enclosing them in brackets (<>).

The following example places the characters m, e, s, s, a, g, e, carriage return, and line feed, in the first nine bytes of the write buffer.

```
IEX$>SET DATA C 0 message<15><12>
```

Individual values can be specified as decimal by appending a decimal point to the value. They can be specified as hexadecimal by appending an X to the value.

The **offset** parameter specifies the offset within the write buffer where the data is to be stored. If the data is specified as word, the offset is also expressed in words. If the data is byte, the offset is byte. The offset is expressed in base 0. Hence, the first location in the write buffer is equivalent to offset 0. **Offset** can be expressed as an octal (default) or decimal number. To specify a decimal offset, append a decimal point to the end of the offset value.

data1 ... datan are the values to be stored in the write buffer. Octal, decimal, and hexadecimal values are separated by spaces. ASCII characters are not separated.

5.2.22 SMD - Set Mode

This command is used to issue the SETMODE command to the driver. A function modifier MUST be specified to indicate the specific mode that is being changed.

Format	Request Generated
SMD!IO$M_BUFFERED 0	IO$_SETMODE!IO$M_BUFFERED
SMD!IO$M_DIRECT 0	IO$_SETMODE!IO$M_DIRECT
SMD!IO$M_TIMOUT 0**seconds**	IO$_SETMODE!IO$M_TIMOUT

The SMD command can be used to select the data path mode (buffered or direct) and to change the number of seconds before a request will timeout. If SMD is used to change the timeout value, the **seconds** parameter specifies the number of seconds before the request will time out. This value must be greater than or equal to two. The timeout period can also be changed with the TMO and STO commands.

The SMD command cannot be used to enable attention ASTs on the occurrence of events. To do this, use the AST command described in Section 5.2.1.

The exerciser waits for this request to complete before continuing.

5.2.23 SNS,SNI - Sense Mode

These commands report the state of an IEX11 unit. The controller state, addressed state, and contents of the IEC/IEEE status register for the unit are displayed. Refer to Section 3.2 for information on the contents of the IEC/IEEE status register.

Format	Request Generated
SNS 0	IO$_SENSEMODE
SNI 0	"

The exerciser waits for this request to complete before continuing.

5.2.24 SPO - Perform a Serial Poll

This command causes the controller-in-charge to perform a serial poll of the selected devices. This command is usually issued after a service request has been detected.

Format	Request Generated
SPO 0 **tlkadr1 [extadr1] ... tlkadrn [extadrn]**	IO$_SER_POLL$

The **tlkadr** parameters specify the talker addresses of the devices to be polled. The optional **extadr** parameters can be used to specify extended addresses. When conducting the serial poll, the driver inspects the byte following the primary address to see if it is a valid secondary address. If it is, the secondary address is also transmitted.

Devices are polled as long as the service request line is asserted. When the serial poll completes, the talker address and serial poll status byte read from the device(s) that requested service are printed.

The exerciser waits for this request to complete before continuing.

5.2.25 TMO,STO - Set Timeout Value

These commands are used to set the length of time before a request to the driver will timeout.

Format	Request Generated
TMO 0 **seconds**	IO$_SETMODE!IO$M_TIMOUT
STO 0 **seconds**	"

The **seconds** parameter specifies the number of seconds (from 2 to 32767.) before a timeout will occur. The default timeout period for the driver is 20. seconds.

The exerciser waits for this request to complete before continuing.

5.2.26 VERIFY - Display Data in Write Buffer

This command is used to display the contents of the internal write buffer. The user can select the format in which the data is displayed. The following options are available:

B - Data displayed as byte values by radix set in SET DATA command
W - Data displayed as word values by radix set in SET DATA command
O - Values displayed in octal format
D - Values displayed in decimal format
X - Values displayed in hexadecimal format
C - Values displayed in ASCII character format

Format	Request Generated
VERIFY [B] [W] [O] [D] [X] [C] **[length]**	None

The optional **length** parameter can be used to specify the number of bytes (or words) of data which will be displayed. **Length** will also default to whatever value is needed to display the last datum entered into the write buffer by the most recent SET DATA command.

5.2.27 WDA,WET,WLB,WLE - Write Data

These commands are used to write data to devices addressed as listeners on the IEC/IEEE bus. The WET and WLE requests cause the EOI line to be asserted when the last byte is written. The WDA and WLB commands output the last byte without asserting EOI.

Format	Request Generated
WDA 0 **bytcnt** or **string**	IO$_WRITELBLK
WLB 0 **bytcnt** or **string**	"
WET 0 **bytcnt** or **string**	IO$_WRITELBLK!IO$M_EOI
WLE 0 **bytcnt** or **string**	"

The data which is written will be contained in either the **string** parameter of the command or the exerciser's internal write buffer. The WFT command can be used to cause the exerciser to wait for the write request to complete.

If the third parameter in the command is a number (**bytcnt**), the exerciser will write that number of bytes from its internal write buffer. The maximum size of the write buffer is 16384. bytes. Data may be entered into the internal write buffer by using the SET DATA command (Section 5.2.21).

If the third parameter in the command is an ASCII string (**string**), the exerciser will output that string and append the IO$M_EOI modifier to the IO$_WRITELBLK request. This form of the command relieves the user of having to use the SET DATA command. Brackets (<>) may be used to specify octal data. To prevent the exerciser from interpreting a character string beginning with a number as a byte count, enclose the string in quotation marks ("). Quotation marks can also be used to output a string which includes embedded spaces.

The exerciser does not wait for this request to complete before continuing.

5.2.28 WFE - Wait for Event

This command causes the exerciser to wait for an event to complete. Recognition of the event MUST previously have been enabled by the REV command. The WFE command displays appropriate event related information when the event occurs.

Format	Request Generated
WFE 0	None

The following exerciser commands would need to be entered to have IXA0: wait for an *addressed as talker* event:

```
IEX$>SEM 0 4  ;Event of addressed as talker
IEX$>REV 0    ;Recognize event
IEX$>WFE 0    ;Wait for event
```

5.2.29 WFQ - Wait for Request

As noted above, the exerciser does not wait for all exerciser commands to complete. The exerciser issues a $QIO request rather than a $QIOW request for some commands. The WFQ and WFT commands cause the exerciser to wait for the previous request on that unit to complete. The WFT command causes the exerciser to wait for the last data transfer request (RDA, RLB, RLM, WDA, WET, WLB, or WLE) to complete. The WFQ command causes the exerciser to wait for any other $QIO request to complete.

Format	Request Generated
WFQ 0	None

5.2.30 WFT - Wait for Transfer

This command causes the exerciser to wait for the last data transfer command on the unit to complete. It then displays the return status code and the number of bytes transferred. For a read request, the type of termination is also indicated.

Format	Request Generated
WFT 0	None

The following termination designations may be displayed for the termination method:

- BYTE COUNT MET - for byte count overflow termination
- EOI DETECTED - for EOI termination
- MATCH CHARACTER - for match character termination

If data checking is enabled (see Section 5.2.6), the data read is compared to the data in the internal write buffer and an error message is generated if any differences are found.

6 DIFFERENCES TO IEC11-V DRIVERS

6.1 INTRODUCTION

The IEC11 is the obsolete predecessor to the IEX11. This chapter is a guide written to help you understand what must take place in order for you to convert programs written for the IEC11 to the IEX11. It can be totally skipped if you are not required to make such conversions.

The drivers for the IEC11 controllers and the IEX11 controller are very similar, and in most cases the user will have little difficulty converting programs from the IEC11 to the IEX11. The more straightforward the programs are, the easier the conversion will be. Most of the differences in the drivers are in minor areas, which will not affect most user programs. Normally, you will be able to make the conversion in a request-by-request manner.

6.2 HARDWARE DIFFERENCES

The IEC11 interface systems contain discrete elements to handle all of the IEC/IEEE bus control and data transfer functions. In contrast to this design, and IEX11 interface contains two identical TMS9914A micro chips, which represent two fully independent IEC/IEEE bus controllers. These chips contain most of the logic which is related to the IEC/IEEE bus. Thus an IEX11 can be viewed as an interface between a VAX processor and two TMS9914A micro chips. The auxiliary commands, in particular, are local commands to these chips. Thus the IEX11 has a completely different hardware design than the IEC11, and this dictates differences in the manner in which the interface is programmed. The register layout of the two interfaces is completely different. Further differences exist in the IEC/IEEE bus status information provided by the interfaces and the method in which events on the IEC/IEEE bus are detected.

The IEX-VMS-Driver was designed to make the most efficient use of the features of the IEX11 interface. This made compatibility with the IEC11-V drivers of secondary importance.

6.3 DIFFERENCES IN DEVICE CHARACTERISTICS

The IEC11 interface to the IEC/IEEE bus consists of two controllers: IEC11-A and IEC11-B. Each controller is serviced by a separate driver, and the two controllers appear as separate devices in the VMS operating system. The IEC11-A has the device mnemonic IEA0: and the IEC11-B has the mnemonic IBA0:. The IEC11-A possesses full IEC/IEEE bus controller capability. The IEC11-A can transfer data but only one byte at a time, with an interrupt being generated after every byte. The IEC11-B can transfer data in DMA mode, but lacks the bus control features of the IEC11-A.

The IEX11 interface incorporates the features of both the IEC11-A and IEC11-B interfaces. It has full bus controller capabilities and is able to transfer data in DMA mode. The IEX11 interface contains two independent bus controllers. These controllers can be connected to the same IEC/IEEE bus or they can be connected to separate buses. These two controllers are fully independent. For instance, if the controllers are connected to different IEC/IEEE buses, they can both be transferring data at the same time. Thus, one IEX11 controller can perform the functions of two IEC11-A and IEC11-B controllers.

The IEC11 interface on the VAX was supported as two devices: IEA0: and IBA0:. For the IEX11 interface, the normal driver generation produces one device with two units: IXA0: and IXA1:.

When the device dependent characteristics are requested through a $GETCHN of $GETDEV system service, the IEX-VMS-Driver returns all of the characteristics returned by the IEC11-V drivers: DEVM_AVL, DEVM_IDV, DEV$M_ODV, and DEV$M_RTM. In addition, IEX-VMS-Driver returns the DEV$M_SHR characteristic indicating that the device is shareable between processes. The IEC11 and IEX11 both have the same device class which is DC$REALTIME. The device type of the IEC11 is undefined, and the device type of the IEX11 is DT$_IX_IEX11.

6.4 GENERAL DRIVER DIFFERENCES

This section describes the general differences between the IEX-VMS-Driver and the IEC11 drivers.

6.4.1 Differences in Entering CACS State

If a unit must be in the CACS state in order to process a request, then both the IEC11-A driver and the IEX-VMS-Driver will attempt to place the unit in this state. If the unit is controller-in-charge, this is done in the same manner in both drivers. The case in which a unit must go to CACS to perform a request when the unit is not controller-in-charge is handled differently in the two drivers. The IEC11-A driver tests whether the unit is system controller, and if this is the case, it sends an interface clear to force the unit to become controller-in-charge. The IEX-VMS-Driver will not make the unit controller-in-charge; it will reject the request with a SS$_ILLIOFUNC error code. This case can only occur in applications in which the controller-in-charge function is passed between devices on the IEC/IEEE bus, since the system controller is always the initial controller-in-charge. If an applications program requires that a unit will regain the controller-in-charge function in this manner, an IO$_INITIALIZE request must be explicitly executed to make the unit controller-in-charge. This difference is not separately noted in the following description of the individual $QIO functions.

6.4.2 Effects of Service Request Interrupts

In the IEC11-A driver, service request (SRQ) interrupts will abort the following functions:

- IO$_PAR_POLL
- IO$_READPBLK
- IO$_REMOTE
- IO$_WATCH
- IO$_WRITEPBLK

The IO$_SETMODE request can be used to enable and disable service request interrupts. The hardware logic of the IEX11 interface will terminate data transfers when any event interrupts occur. For this reason the IEX-VMS-Driver, by default, disables all event interrupts before data transfer, and then re-enables the same interrupts after the transfer has completed. Thus, when using the IEX-VMS-Driver, there is generally no need to disable service request interrupts once they have been enabled. Other functions of the IEX11 are not affected by event interrupts.

When converting from the IEC11 interface to the IEX11 interface, it is not sufficient to simply convert the requests which enable and disable service request interrupts; the effects of the interrupts themselves must be taken into account. If the logic of a program using the IEC11-V drivers relies upon the fact that service requests can terminate data transfers, then the function modifier IO$M_ENA_SRQ must be inserted in the data transfer requests when the program is converted to the IEX11.

6.5 I/O FUNCTION DIFFERENCES

The following table presents a summary of the differences in the $QIO calls of the two drivers. It is intended to indicate where changes must be made when converting a program from the IEC11-V drivers to the IEX-VMS-Driver. The column referring to the functionality of the $QIO assumes a typical application. When the same functionality is indicated for both drivers, it means that any differences are so minor they will not present problems in the typical application. It does not mean that the $QIOs are necessarily 100% identical in all applications. The functionality indication also assumes that any differences in the parameter format or modifiers have been converted by the user.

Table 6–1 Summary of IEC11 I/O Function Differences

IEC11 Function	Equivalent IEX11 Function	Different Parameters	Different Modifiers	Different Functionality
IO$_INITIALIZE	yes	yes	yes	no
IO$_PAR_POLL	yes	no	no	no
IO$_READVBLK	yes	no	yes	no
IO$_RELEASE	yes[1]	yes	no	no
IO$_REMOTE	no[4]	–	–	–
IO$_SENSEMODE	yes	no	no	no[5]
IO$_SER_POLL	yes	no	no	no
IO$_SETMODE	yes	yes	yes	yes
IO$_WATCH	no[4]	–	–	–
IO$_WRITEVBLK	yes	no	yes	no[2]
IO$_WRITEVBLK!IO$M_COMMAND	yes	no	no	no[3]

[1]The IO$_INITIALIZE function should be used as the equivalent function.

[2]Refers to data transfers.

[3]Requires that the IEX-VMS-Driver IO$_COMMANDS function be used.

[4]Function can be realized through other IEX-VMS-Driver functions.

[5]The information returned and its format are different.

6.5.1 IO$_INITIALIZE - Device Initialization

For both the IEC11-V drivers and IEX-VMS-Driver, the IO$_INITIALIZE function is used to initialize the device at the start of a process. For both drivers this function is used to specify if the unit is to be system controller and if the unit is to be controller-in-charge. For the IEC11-V driver, these characteristics are specified through function modifiers. For the IEX-VMS-Driver, bits in the mask parameter in P1 are used.

IEC11-V Drivers	IEX-VMS-Driver
IO$M_SACS modifier	Bit 2 in mask parameter
IO$M_CACS modifier	Bit 4 in mask parameter

To convert an IEC11 function to an IEX11 function, the function modifiers should be removed from the function and the P1 mask parameter should be added to the $QIO. This mask parameter contains six significant bits which must be specified. If the unit is to be system controller and controller-in-charge, then a value of 67_8 should be specified for this parameter. If the unit is not to be system controller and is not to be controller-in-charge, then a value of 43_8 should be specified. In the second parameter P2, the IEC/IEEE bus address for the unit must be specified.

Example 6–1 Initialize Unit As System Controller and Controller-in-charge

IEC11 $QIO:

```
$QIOW_S   FUNC=IO$_INITIALIZE!IO$M_SACS!IO$M_CACS, -
          .
          .
          .
```

IEX-VMS-Driver $QIO:

```
MASK = ^O67
$QIOW_S   FUNC=IO$_INITIALIZE, -
          .
          .
          .
          P1 = MASK,-      ;Initialization mask, octal 67
          P2 = #0          ;IEC/IEEE bus address
```

Note that P1 requires a different addressing mode than the other parameters.

6.5.2 IO$_PAR_POLL - Perform a Parallel Poll

The parallel poll function is identical in the IEC11-V drivers and IEX-VMS-Driver and requires no conversion.

6.5.3 IO$_READPBLK - Read Data

The IEC11-V drivers and IEX-VMS-Driver read data in essentially the same manner. There are, however, a few minor differences in the implementation of this function in the drivers.

The IEX-VMS-Driver does not require the IO$M_EOI modifier, since the driver enables both EOI and byte count overflow interrupts when the request is processed. The IO$M_MATCH modifier is also not required by the IEX-VMS-Driver. If P3 is nonzero, then the driver uses P3 and P4 to enable match count termination. When the function is complete, a code indicating the type of termination is returned in the status buffer. Thus when converting read requests, the IO$M_EOI and IO$M_MATCH modifiers should simply be deleted.

In both the IEC11-V drivers and the IEX-VMS-Driver, the three read functions: IO$_READVBLK, IO$_READLBLK, and IO$_READPBLK are considered to be equivalent functions, as long as the issuing process has logical I/O privilege.

There is no equivalent in the IEX-VMS-Driver to the IO$M_REQSRV function modifier of the IEC11-B driver. This modifier instructs the driver to make a service request at the end of the data transfer. If this feature is required, an IO$_SERVICE $QIO must be inserted after the write request when the programs are converted for the IEX-VMS-Driver.

There is also no equivalent of the IO$M_NOCONSEQ function in the IEX-VMS-Driver. This modifier enables nonconsecutive match termination, and the IEX11 hardware can only detect consecutive match termination in DMA mode. If this feature is required, the function must be replaced by multiple reads, which specify a one character match termination.

6.5.4 IO$_RELEASE - Device Release

This function is used in the IEC11-V drivers to release a unit so that it can be used by another process. Since the IEX-VMS-Driver treats units as shareable devices, this function is not required. For the IEC11-A and IEC11-B drivers this function resets the device dependent status. For the IEC11-A driver a master clear is performed on the interface, and if the unit is system controller, and IEC/IEEE interface clear is sent. When converting programs to the IEX11 interface, the IO$_RELEASE $QIOs should be replaced by IO$_INITIALIZE $QIOs. Bit zero in the mask parameter (P1) of the IO$_INITIALIZE request specifies that a master clear is to be made on the interface. If bit four of the mask parameter is set, an interface clear will be sent. The IO$_INITIALIZE request automatically resets the status of the unit.

6.5.5 IO$_REMOTE - Set Remote Enable

This function is used to place other devices on the IEC/IEEE bus into a remote state. The IEX-VMS-Driver requires two requests to realize this operation. First the auxiliary command **sre** (220_8) must be issued, and then the selected devices must be addressed as listener.

To convert an IEC11 IO$_REMOTE function for the IEX-VMS-Driver, the function code IO$_REMOTE should be replaced by the IO$_COMMANDS function code. Immediately before this $QIO, an IO$_AUXILIARY function must be added, with a value of 220_8 as the P1 parameter.

Devices are taken out of the remote state by sending the **sre** auxiliary command with bit seven clear (20_8).

6.5.6 IO$_SENSEMODE - Sense Mode

The IEC11-V drivers process this function by composing a status word and returning it in the second word of the I/O status block. This status word has the following format:

Table 6–2 Format of Status Word In IEC11 I/O Status Block

Bit	Meaning
0	System controller. This information is not returned by the IEX-VMS-Driver IO$_SENSEMODE function. Your program, however, knows if the unit is system controller, since this is a parameter of the IO$_INITIALIZE request.
1	IEC/IEEE NRFD line. This bit corresponds to bit 12 of the IEC/IEEE status register.
2	LON switch. This information is not returned by the IEX-VMS-Driver IO$_SENSEMODE function. The IEX11 can only address itself as listener through a **lon** auxiliary command, so that if the unit is controller-in-charge and addressed as listener, then **lon** is enabled.
3	not used
4	not used
5	not used
6	IEC/IEEE SRQ line. This bit corresponds to bit 10 of the IEC/IEEE status register.
7	IEC/IEEE REN line. This bit corresponds to bit 8 of the IEC/IEEE status register.
8	Bus controller active. The controller state of the unit is returned by the IEX-VMS-Driver IO$_SENSEMODE request in the low four bits of the fourth word of the I/O status block.
9	Bus controller in standby. The controller state of the unit is returned by the IEX-VMS-Driver IO$_SENSEMODE request in the low four bits of the fourth word of the I/O status block.
10	Parallel poll in progress. This information is not provided by the IEX11 interface.
11	Talker is active. When this bit is set, it is the equivalent of the unit being addressed as talker and the attention line being reset. The state of the attention line is presented in bits 5 and 15 of the IEC/IEEE status register. The addressed state of the unit is returned in the high byte of the fourth word of the I/O status block.
12	Listener is active. When this bit is set, it is the equivalent of the unit being addressed as listener and the attention line being reset. See definition for bit 11.
13	Interface is being cleared. This bit corresponds to bit 9 of the IEC/IEEE status register.
14	Remote enable state active. This bit corresponds to bit 7 of the IEC/IEEE status register.
15	Device is being polled. This information is not provided by the IEX11 interface.

In the IEX-VMS-Driver the IO$_SENSEMODE function is not queued to the driver; rather it is immediately processed in the I/O packet preprocessing code of the driver. This means that this function will be immediately processed, even if the driver is currently processing another function. Normally this feature need not be considered when converting programs for the IEX-VMS-Driver. If it is necessary to have the IO$_SENSEMODE functions queued to the driver, then a dummy function modifier should be specified.

The status information returned by the IEX-VMS-Driver for this function is identical to the information returned in the I/O status block after every IEX-VMS-Driver function. The contents of the IEX11 IEC/IEEE status register is returned in the third word of the I/O status block returned by the driver.

6.5.7 IO$_SER_POLL - Perform a Serial Poll

A serial poll is issued by the controller-in-charge to determine the devices on the IEC/IEEE bus that are requesting service. This function is processed in the IEC11-A driver and IEX-VMS-Driver in the same manner, and the parameter format is identical in both drivers.

There are only minor differences in the processing of this function in the two drivers. The IEC11-A driver tests the SRQ line after a device is polled, while the IEX-VMS-Driver tests this line before a device is polled. This means that the IEC11-A driver will always poll at least one device, whereas the IEX-VMS-Driver will not poll any devices if the SRQ line is not set and the IO$M_ALLDEVICES modifier is not specified.

The IEC11-A driver blocks service request interrupts during the processing of the serial poll. This is not done in the IEX-VMS-Driver. The contents of the IEC/IEEE status register returned in the I/O status block should always be checked for an outstanding SRQ following a serial poll.

The IEC11-A driver will return an SS$_ABORT error code if at the end of a serial poll the SRQ line is set. This test is not made in the IEX-VMS-Driver. The user can, however, test the SRQ bit in the IEC/IEEE status register which is returned in the third word of the I/O status block.

6.5.8 IO$_SETMODE - Set Mode

The IO$_SETMODE function is used in both drivers to perform miscellaneous utility functions. A function modifier must always be present to select the function to be performed. In both of the drivers only one of the possible functions can be performed per $QIO call. The following table shows a summary of the function modifiers for the IEC11 IO$_SETMODE function.

IEC11-V IO$_SETMODE Modifier	Meaning
IO$M_CLR_REN	Clear remote enable
IO$M_DIS_SRQ	Disable SRQ interrupts
IO$M_ENA_SRQ	Enable SRQ interrupts
IO$M_IFC	Send interface clear
IO$M_SERVICE	Make a service request
IO$M_TIMOUT	Specify timeout duration
IO$M_UNS_AST	Set unsolicited AST address

6.5.8.1 IO$M_CLR_REN - Clear Remote Enable

The IEC11-A driver used the IO$_REMOTE function to place devices in a remote state. In this state devices will accept commands from the IEC/IEEE bus. The IO$_SETMODE!IO$M_CLR_REN request is used to reset the devices to the local state. In this state the devices are under local manual control.

For the IEX-VMS-Driver a different sequence is used. Devices are set to the remote state by issuing the **sre** (220_8) auxiliary command and then addressing the devices as listener. All devices are returned to the local state through the **sre** (20_8) auxiliary command.

When converting programs, all IO$_SETMODE!IO$M_CLR_REN $QIO calls should be replaced by IO$_AUXILIARY function calls with P1 equal to 20_8.

6.5.8.2 IO$M_DIS_SRQ - Disable Service Request Interrupts

This modifier is used to disable service request interrupts. It is the complement of the IO$M_ENA_SRQ request. If this request was used to disable service request interrupts during a data transfer, then the $QIO call can be deleted when converting programs, since the IEX-VMS-Driver will automatically disable all event interrupts for the duration of the data transfer. Otherwise, an IO$_SETEVENT $QIO call with bit zero of P1 cleared can be used to disable service request interrupts.

6.5.8.3 IO$M_ENA_SRQ - Enable Service Request Interrupts

This modifier is used to enable service request interrupts in the IEC11-A driver. If a service request interrupt then occurs during a data transfer, the transfer will be aborted with the SS$_ABORT error code. If a service request interrupt occurs when the unit is not transferring data, an unsolicited AST will be delivered, if one had been previously established by the program.

These two cases are handled separately in the IEX-VMS-Driver. The IO$_SETEVENT function is used to enable service request interrupts for the purpose of detecting normal service requests. These service requests will also generate attention ASTs if an IO$_SETMODE!IO$M_ATTNAST function has been issued to specify an AST address. Normally the driver disables all event related interrupts during data transfers, so that even if service request interrupts are enabled by the IO$_SETEVENT function, a data transfer will not be interrupted by a service request. If it is required that service requests be able to interrupt data transfer, then the IO$M_ENA_SRQ modifier must be used with the data transfer $QIOs. This modifier enables service request interrupts only for the duration of the data transfer. These two methods of enabling service request interrupts are independent. This means that service request interrupts can be enabled with the IO$M_ENA_SRQ modifier without previously being enabled through an IO$_SETEVENT function.

Programs for the IEC11 interface will normally have enabled service request interrupts in order to detect the occurrence of service requests. For these cases the IO$_SETMODE!IO$M_ENA_SRQ $QIO calls should be replaced by an IO$_SETEVENT call with bit zero of the first parameter (P1) set. Care must be taken that the other bits in this parameter reflect the events for which recognition is to be enabled. For example, if a previous IO$_SETEVENT function was issued to enable other events, then an IO$_SETEVENT $QIO call with only bit zero set would disable the recognition of the events specified in the first call.

6.5.8.4 IO$M_IFC - Send Interface Clear

This modifier is used to send the interface clear message across the IEC/IEEE bus. This message may only be sent by the device that is system controller. Sending the interface clear message makes the system controller controller-in-charge. It is also used to initialize other devices on the IEC/IEEE bus. In the IEX-VMS-Driver an interface clear is automatically sent in the IO$_INITIALIZE function, when the mask parameter specifies that the unit is to be controller-in-charge. Independent thereof, your program can generate an interface clear through the **sic** (217_8) auxiliary command. An IO$_AUXILIARY $QIO call should first be issued with P1 equal to 217_8. Immediately thereafter a second IO$_AUXILIARY $QIO call should be issued with P1 equal to 17_8. The IEX11 hardware requires that the delay time between the 217_8 and the 17_8 auxiliary commands be at least 100 microseconds, but since this time is shorter that the $QIO overhead, your program will require no delay.

In converting programs to the IEX-VMS-Driver, the IO$_SETMODE!IO$M_IFC function should be replaced by two IO$_AUXILIARY requests with P1 values of 217_8 and 17_8.

6.5.8.5 IO$M_SERVICE - Make A Service Request

This modifier is used to make a service request. In the IEX-VMS-Driver a separate function, IO$_SERVICE, is used to make a service request. The IO$_SERVICE function expects the status byte in P1. The IEC11-V drivers expect the status byte in P3. Thus to convert this request to the IEX-VMS-Driver, the function code IO$_SETMODE!IO$M_SERVICE should be replaced by IO$_SERVICE, and the status byte should be moved from P3 to P1.

The IEC11-V drivers disable the normal timeout logic during this request, so that if the unit is not addressed in a serial poll, the request will hang for 65,536 seconds. The IEX-VMS-Driver uses the standard timeout logic for this request, and as a result you can specify how long the driver should wait for a serial poll.

The service request function in the IEX-VMS-Driver contains two features that are not present in the IEC11-V drivers. If a service request is issued with bit six of the status byte cleared, then the IEX-VMS-Driver will load the status byte into the serial poll register, but the driver will not wait to be addressed in a serial poll. Since bit six is cleared, such a request will not cause a service request. If a nonzero value is present in P2, the driver treats this as an extended address. If P2 contains a valid secondary address, the driver will require the unit to receive both the specified primary and secondary addresses before it will respond to a serial poll.

6.5.8.6 IO$M_TIMOUT - Set Timeout Duration

Both the IEC11-V drivers and the IEX-VMS-Driver use this modifier to alter the timeout duration. The IEC11-V drivers expect the timeout duration in P3 while the IEX-VMS-Driver expects this value in P1. In all drivers the timeout duration is expressed in seconds. All drivers have a default duration of 20 seconds. In converting programs, the only modification required for these requests is that the seconds value be moved from P3 to P1.

There are few minor differences in the timeout logic in the drivers. The IEC11-V drivers have a fixed unalterable timeout duration of two seconds for receiving the status byte from a device in a serial poll. When waiting to be addressed in a serial poll in the service request function, the timeout period is always 65,536 seconds. In both cases the IEX-VMS-Driver uses the standard timeout duration. When waiting for a state change before a READ or a WRITE is initiated, the IEC11-V drivers also have an unalterable time duration of 65,536 seconds. For this case the IEX-VMS-Driver uses the standard timeout duration.

6.5.8.7 IO$M_UNS_AST - Set Unsolicited AST Address

This modifier is used to establish the address of the AST routine, which is entered when selected events occur of the IEC/IEEE bus. The IEX-VMS-Driver uses the standard VMS attention ASTs to realize this feature. This presents the following differences in programming this feature for the IEX-VMS-Driver:

1. The IO$_SETMODE modifier to enable ASTs is the standard VMS Modifier IO$M_ATTNAST, rather than the IEC11 modifier IO$M_UNS_AST.
2. The address of the AST routine is specified in parameter P1 and not in the AST parameter of the $QIO request. The AST parameter in the $QIO has the standard VMS usage.
3. The event flag in the $QIO call has the normal VMS usage. It is not set when an attention AST is delivered.
4. Only one attention AST is delivered. Attention ASTs must be reenabled with a new IO$_SETMODE!IO$M_ATTNAST function.

In addition, there are differences in the manner in which events are enabled and in how they are reported. These differences are explained further in Section 6.5.10.

6.5.8.8 IO$_WATCH - Transfer Watch

This modifier has no counterpart in the IEX-VMS-Driver. The transfer watch function is used to detect the end of a data transfer between two other devices on the bus. If a program that uses this function must be converted, then the program must be rewritten to participate in the data transfer. This would involve replacing the IO$_WATCH requests with read data requests.

Another method used to signal the end of data transfer between two devices, neither of which are controller-in-charge, is to have one of the devices make a service request at the end of the transfer. The controller-in-charge will detect the service request and re-assert control over the bus.

6.5.9 IO$_WRITEPBLK - Write Data

In both the IEC11-V drivers and the IEX-VMS-Driver, the three write functions: IO$_WRITEVBLK, IO$_WRITELBLK, and IO$_WRITEPBLK are considered to be equivalent functions, as long as the issuing process has logical I/O privilege. The IEC11-B driver is able to transfer data only. The IEC11-A driver uses the write function to transfer both IEC/IEEE bus commands and data. In the IEX-VMS-Driver the transmission IEC/IEEE bus commands and the transmission of data are treated as two different functions.

6.5.9.1 Transferring IEC/IEEE Bus Commands

In the IEC11-A driver the IO$_COMMAND modifier is used to indicate that the specified buffer contains IEC/IEEE bus commands rather than data. In the IEX-VMS-Driver a separate function, IO$_COMMANDS), is used to transfer commands. To convert the $QIO calls which transfer IEC/IEEE bus commands, the IO$_WRITEPBLK!IO$M_COMMAND) function code should be replaced by the IO$_COMMANDS function code. The request parameters are identical in both drivers.

Both drivers have the function modifier IO$M_STANDBY to instruct the driver to place the unit in CSBS after the commands have been transferred.

6.5.9.2 Transferring Data

The IEC11-V drivers and IEX-VMS-Driver transfer data in essentially the same manner. In most cases these requests will require no conversion.

There are, however, a few minor differences in the implementation of this function in the drivers. There is no equivalent in the IEX-VMS-Driver to the IO$M_REQSRV function modifier of the IEC11-B driver. This modifier instructs the driver to make a service request at the end of the data transfer. If this feature is required, an IO$_SERVICE $QIO must be inserted after the write request when the programs are converted for the IEX-VMS-Driver.

The IEC11-B driver uses P3 to signal whether the EOI line should be asserted during the transmission of the last byte. In the IEX-VMS-Driver, EOI termination can only be selected through the IO$M_EOI function modifier.

The IEC11-A driver has the ability to send consecutive match characters after the specified data buffer has been transferred. This is not possible with the IEX-VMS-Driver, since all data is transferred by means of DMA. When match characters must be sent, they must be included in the data buffer.

6.5.10 Unsolicited AST Mechanism

The IEC11-V drivers use the unsolicited AST mechanism to report events on the IEC/IEEE bus. The AST mechanism is enabled through an IO$_SETMODE!IO$M_UNS_AST $QIO call. When one of the possible events occurs, the driver will queue an AST for the user's process. When the user process is restarted, control will be given to the AST routine. This routine can either initiate event dependent processing, or signal the primary process logic that an event occurred.

The IEX-VMS-Driver contains two methods for recognizing IEC/IEEE bus events. The driver can be programmed to generate unsolicited ASTs as in the IEC11-V drivers, and events can also be recognized through the IO$_REC_EVENT function. For most applications it will be easier to retain the unsolicited AST method of detecting events. If the problem related to the special functions described below is encountered, it may be advantageous to use the IO$_REC_EVENT function. The method of establishing the address of the AST routine and the manner in which the AST is delivered is the same in all drivers. The event number is always delivered in the AST parameter, but the numbers assigned to events differ between the IEC11-V and the IEX-VMS-Driver.

In the IEC11-V drivers, all events (Table 6–3) with the exception of service requests are always enabled. A program can selectively enable and disable service request interrupts. With the IEX-VMS-Driver the user can selectively enable events. Before any events can occur, the selected events must be enabled through the IO$_SETEVENT function. The mask parameter of this request selects the events for which detection is desired. Event detection should be disabled when a process exits by issuing an IO$_SETEVENT $QIO with the mask parameter equal to zero.

Another feature in the IEX-VMS-Driver which is not present in the IEC11-V drivers is the DAC (data accepted) holdoff. When one of the IEX-VMS-Driver events occurs, the bus will be locked. This is done to give the program time to respond to the event. After the program has performed any event dependent processing that may be required, the IEC/IEEE bus must be released by a **dacr** auxiliary command. The driver will note that a holdoff release is pending, and if a subsequent request other than IO$_SENSEMODE is made to the driver for the affected unit, the driver will automatically generate a holdoff release. If the program does not issue a new request to the driver after one of the indicated events has occurred, then the **dacr** disable auxiliary command must be issued through the IO$_AUXILIARY function.

Due to hardware incompatibilities between the interfaces, the events reported are different. The IEC11-V drivers are able to recognize the following events:

Table 6–3 IEC11-V Driver Events

AST Parameter Value	Event
1	State change to TACS
2	State change to LACS
3	State change to SPAS (IEC11-A only)
4	State change to CACS (IEC11-A only)
5	State change to CIDS (IEC11-A only)
6	State change to CSBS (IEC11-A only)
7	Service request (IEC11-A only)
8	Hardware error condition (IEC11-A only)
9	Unrecognized interrupt (IEC11-A only)

The IEX-VMS-Driver can detect the following events:

Table 6–4 IEX-VMS-Driver Events

AST Parameter Value	Event
1	Service request
2	Addressed as listener[1]
3	Addressed as talker[1]
4	Deaddressed
5	Addressed as extended listener[1]
6	Addressed as extended talker[1]
7	Device clear[1]
8	Group execute trigger[1]
9	Remote/local change[1]
10	Received control[1]
11	Parallel poll configure[1]
12	Parallel poll unconfigure[1]
13	Interface clear

[1]Generates a data accepted holdoff

In the majority of applications in which a computer is interfaced to an IEC/IEEE bus, the computer acts as system controller and controller-in-charge. In this role the only event that need be detected is a service request. The IEC11-A and the IEX-VMS-Driver deliver this event in a similar manner, and for these applications the conversion effort will be minimal. The programs must only be changed to test for a value of 1 in the AST parameter, instead of a value of 7.

A typical case in which a computer is not controller-in-charge is when there is another computer or intelligent device on the IEC/IEEE which assumes this role. In such applications, it is necessary that programs can detect when the interface is addressed as talker or listener by the controller-in-charge. The IEC11 interfaces can generate interrupts when the interface enters the TACS or LACS states. These states mean that the interface is addressed as talker (TACS) or listener (LACS) and that the attention line is low. The controller-in-charge sets the attention line low so that data, as opposed to commands, can be transmitted across the IEC/IEEE bus. Thus a transition to TACS or LACS is an indication that the unit should transmit or receive data. The IEX11 interface can detect that it is addressed as talker or listener, but these events are not conditioned upon the state of the attention line. In particular, no interrupt is generated by the IEX11 when the state of the attention line changes. In most cases this difference will not be critical for program conversion, and the IEX11 *addressed as talker* and *addressed as listener* events can be used as the equivalent of the state change to TACS and LACS events.

In both the IEC11-B and IEX-VMS-Driver, when the unit is not controller-in-charge, the driver will wait to be addressed as listener for read data requests or as talker for write data requests, before initiating the DMA transfer. After being addressed in this case, the IEC11-B driver will generate a state change to LACS or state change to TACS event. The IEX-VMS-Driver will automatically generate an ACDS holdoff release so that the IEC/IEEE bus is not locked and the data transfer can proceed.

There are, however, a few cases in the IEC/IEEE environment when a device will be addressed as listener by the controller-in-charge and not be expected to receive data. This occurs when the controller-in-charge addresses the device as listener for one of these special functions:

- selective device clear
- device trigger
- remote/local change

If these functions are mixed in an application with data reception requests, then special logic may be required for a process to determine when it is being addressed as listener for one of these functions, and when it is being addressed as listener so that it can receive data. For these special functions, the IEX11 hardware first generates an *addressed as listener* event and then the event related to the function. If a process receives an *addressed as listener* event, it must wait until either one of the special events is detected or the attention line is set low (the ATN bit in the IEC/IEEE status register is cleared). Both of these conditions can be tested for with one $QIO call. An IO$_REC_EVENT with the IO$M_NOWAIT modifier will return an event if one has occurred, or the SS$_MSGNOTFND status code if none has occurred.

As with every request, the contents of the IEC/IEEE status register is returned in the third word of the I/O status block. Bits 5 and 15 of this word indicate the state of the attention line. The two bits always have the same state, so only one need be tested. For this method to function, the special events (*device clear, device trigger,* and *remote/local change*) must be enabled through an IO$_SETEVENT request.

A state change to CACS occurs when the controller-in-charge function is transferred to the interface. This is equivalent to the received control event in the IEX-VMS-Driver, and the programs must only be converted to recognize an event code of 10 instead of 4.

A state change to CIDS event occurs when the interface has been given the controller-in-charge function, and another device which is the system controller sends the interface clear message. This will make that device controller-in-charge and force all other devices on the IEC/IEEE bus into the idle state. The IEC11-A state change to CIDS event is equivalent to the IEX11 *interface clear* event, and the programs must only be converted to recognize an event code of 13 instead of 5.

All of the other IEC11 events cannot be detected by the IEX11 interface.

The following table summarizes the changes to be made when converting the unsolicited AST mechanism from the IEC11 to the IEX11 interface.

IEC11 Event	IEX11 Event	(IEX11) Event Code
State change to TACS	Addressed as talker	3
State change to LACS	Addressed as listener[1]	2
State change to SPAS	No comparable event	n/a
State change to CACS	Received control	10
State change to CIDS	Interface clear	13
State change to CSBS	No comparable event	n/a
Service request	Service request	1
Hardware error condition	No comparable event	n/a
Unrecognized interrupt	No comparable event	n/a

[1]May require special processing if the events device clear, device trigger, or remote/local change can occur

A EXAMPLE OF IEX EXERCISER EXECUTION USING IEX$DEMO.COM

This appendix contains a sample output of the IEX Exerciser which is included in the kit. Refer to Chapter 5 for a description of the exerciser commands.

```
$SET DEFAULT SYS$SYSROOT:[SYSHLP.EXAMPLES.IEX]
$RUN IEX
Copyright (C) Digital Equipment Corporation. 1989, 1990. All Rights Reserved.
IEX$>@IEX$DEMO
IEX$>; .TITLE IEX$DEMO - IEX$DEMO.COM demonstration command file
IEX$>; .IDENT /IEX-VMS V4.2/
IEX$>;
IEX$>;------------------------------------------------------------------------;
IEX$>;                            IEX$DEMO.COM                               ;
IEX$>;                        IEX-VMS-Driver V4.2                            ;
IEX$>;------------------------------------------------------------------------;
IEX$>;
IEX$>; COPYRIGHT (C) 1986,1987,1988,1989,1990 DIGITAL EQUIPMENT CORP.,
IEX$>; COMPUTER SPECIAL SYSTEMS - I.S.G.
IEX$>; MERRIMACK, N.H. 03054
IEX$>;
IEX$>; Original COPYRIGHT (C) 1983,1984,1985,1986 DIGITAL EQUIPMENT GMBH,
IEX$>;     COMPUTER SPECIAL SYSTEMS
IEX$>;     MUNICH, FEDERAL REPUBLIC OF GERMANY
IEX$>;
IEX$>;--
IEX$>;
IEX$>; INITIALIZE BOTH CONTROLLERS SETTING IXA0: TO CONTROLLER IN CHARGE.
IEX$>; IXA0: IS ASSIGNED THE IEEE-488 BUS ADDRESS OF 0 AND IXA1: IS
IEX$>; ASSIGNED THE IEEE-488 BUS ADDRESS OF 1.
IEX$>;
IEX$>INI 0 67 0
IEX$>INI 1 43 1
IEX$>;
IEX$>STO 0 20.        ;SET TIME OUT TO STANDARD VALUE  20 SECONDS
IEX$>STO 1 20.        ;SET TIME OUT TO STANDARD VALUE  20 SECONDS
IEX$>;
IEX$>; TEST THAT THE VECTOR ADDRESS SELECTED ON THE INTERFACE
IEX$>; CORRESPONDS TO THE VECTOR SPECIFIED IN THE DRIVER.
IEX$>; IF THE INTERRUPT VECTOR IS NOT CORRECT, THE FOLLOWING
IEX$>; WFQ REQUEST WILL TIME OUT AFTER 20 SECONDS.
IEX$>; THIS SEQUENCE ALSO TESTS THAT THE TWO CONTROLLERS
IEX$>; ARE CONNECTED TOGETHER.
IEX$>;
IEX$>CMD 0 41        ;THIS COMMAND GENERATES AN INTERRUPT
IEX$>WFQ 0           ;WAIT FOR QIO TO COMPLETE
IEX$>CMD 0 77        ;UNLISTEN
IEX$>;
IEX$>;
IEX$>; DEMONSTRATE THE VARIOUS IEE-488 CONTROLLER AND DEVICE STATES
IEX$>;
IEX$>; SENSE THE STATE OF IX: AND IXA1:
IEX$>;
IEX$>SNS 0           ; EXPECTED: CACS, NOT ADDRESSED
    CONTROLLER ACTIVE STATE
    NOT ADDRESSED
    IEEE STATUS REGISTER: 120040
IEX$>SNS 1           ; EXPECTED: CIDS, NOT ADDRESSED
    CONTROLLER IDLE STATE
    NOT ADDRESSED
    IEEE STATUS REGISTER: 120041
```

EXAMPLE OF IEX EXERCISER EXECUTION USING IEX$DEMO.COM

```
IEX$>;
IEX$>; SET IXA0: TO STANDBY
IEX$>;
IEX$>GTS 0
IEX$>;
IEX$>; IXA0: SHOULD NOW BE IN CONTROLLER STANDBY STATE AND THE ATTENTION
IEX$>; LINE BITS (BITS 5 & 15) IN THE STATUS REGISTER SHOULD BE RESET
IEX$>;
IEX$>SNS 0           ;EXPECTED: CSBS, NOT ADDRESSED
    CONTROLLER STANDBY STATE
    NOT ADDRESSED
    IEEE STATUS REGISTER:      0
IEX$>;
IEX$>; NOW HAVE IXA0: GET CONTROL
IEX$>;
IEX$>GCT 0
IEX$>;
IEX$>; IXA0: SHOULD NOW BE IN THE CONTROLLER ACTIVE STATE AND THE
IEX$>; ATTENTION LINE BITS (5 & 15) IN THE STATUS REGISTER SHOULD
IEX$>; BE SET.
IEX$>;
IEX$>SNS 0           ;EXPECTED: CACS, NOT ADDRESSED
    CONTROLLER ACTIVE STATE
    NOT ADDRESSED
    IEEE STATUS REGISTER: 120040
IEX$>;
IEX$>; A DEVICE CAN HAVE THREE ADDRESSED STATES:
IEX$>;
IEX$>;     1. NOT ADDRESSED
IEX$>;     2. ADDRESSED AS LISTENER
IEX$>;     3. ADDRESSED AS TALKER
IEX$>;
IEX$>; IN ORDER TO RECEIVE DATA A DEVICE MUST BE ADDRESSED AS
IEX$>; LISTENER.  IN ORDER TO SEND DATA A DEVICE MUST BE ADDRESSED
IEX$>; AS TALKER.
IEX$>;
IEX$>; ONLY THE CONTROLLER IN CHARGE CAN ADDRESS OTHER DEVICES
IEX$>; ON THE IEEE-488 BUS.  LISTENER ADDRESSES ARE IN THE RANGE
IEX$>; 40-76 AND TALKER ADDRESSES ARE IN THE RANGE 100-136,
IEX$>; WHERE THE LOW ORDER FIVE BITS ARE THE DEVICE ADDRESS.
IEX$>; CODE 77 WILL UNLISTEN ALL DEVICES, AND CODE 137 WILL
IEX$>; UNTALK ALL DEVICES. IN ORDER TO ADDRESS ANOTHER DEVICE
IEX$>; THE RESPECTIVE COMMAND MUST BE TRANSFERRED VIA THE
IEX$>; COMMAND OUTPUT REGISTER.
IEX$>;
IEX$>CMD 0 41        ;ADDRESS IXA1: AS LISTENER
IEX$>SNS 1           ;EXPECTED: CIDS, LISTENER
    CONTROLLER IDLE STATE
    ADDRESSED AS LISTENER
    IEEE STATUS REGISTER: 120065
IEX$>CMD 0 77        ;UNLISTEN ALL DEVICES
IEX$>SNS 1           ;EXPECTED: CIDS, NOT ADDRESSED
    CONTROLLER IDLE STATE
    NOT ADDRESSED
    IEEE STATUS REGISTER: 120041
IEX$>;
IEX$>CMD 0 101       ;IXA1: TO TALKER
IEX$>SNS 1           ;EXPECTED: CIDS, TALKER
    CONTROLLER IDLE STATE
    ADDRESSED AS TALKER
    IEEE STATUS REGISTER: 120053
IEX$>CMD 0 5FX       ;UNTALK (TERMINATING A NUMBER WITH X INDICATES HEXADECIMAL)
IEX$>SNS 1           ;EXPECTED: CIDS, NOT ADDRESSED
    CONTROLLER IDLE STATE
    NOT ADDRESSED
    IEEE STATUS REGISTER: 120041
IEX$>;
IEX$>; MANY MEASURING DEVICES MUST BE PUT INTO A REMOTE
IEX$>; STATE BEFORE THEY WILL ACCEPT DATA FROM THE IEEE
IEX$>; BUS.  DEVICES ARE SET TO REMOTE BY ISSUING THE
```

```
IEX$>; REMOTE ENABLE AUXILIARY COMMAND AND THEN ADDRESSING
IEX$>; THE DEVICES AS LISTENER.
IEX$>;
IEX$>AUX 0 220       ;REMOTE ENABLE
IEX$>CMD 0 41 77     ;ADDRESS AS LISTENER
IEX$>;
IEX$>;
IEX$>; IF THE DRIVER MUST CHANGE STATES IN ORDER TO COMPLETE A
IEX$>; REQUEST, THEN IT WILL AUTOMATICALLY RESET THE CONTROLLER
IEX$>; TO THE ORIGINAL STATE AFTER THE REQUEST COMPLETES.
IEX$>;
IEX$>GTS 0           ;GO TO STANDBY STATE
IEX$>SNS 0           ;EXPECTED: CSBS, NOT ADDRESSED
    CONTROLLER STANDBY STATE
    NOT ADDRESSED
    IEEE STATUS REGISTER:    400
IEX$>;
IEX$>; IN ORDER TO TRANSMIT A COMMAND THE CONTROLLER MUST GO TO
IEX$>; CACS.
IEX$>;
IEX$>CMD 0 41        ;ADDRESS IXA1: AS LISTENER
IEX$>SNS 1           ;EXPECTED: CIDS, LISTENER
    CONTROLLER IDLE STATE
    ADDRESSED AS LISTENER
    IEEE STATUS REGISTER:  20625
IEX$>SNS 0           ;EXPECTED: CSBS, NOT ADDRESSED
    CONTROLLER STANDBY STATE
    NOT ADDRESSED
    IEEE STATUS REGISTER:    400
IEX$>GCT 0           ;GO TO CACS
IEX$>CMD 0 77        ;UNLISTEN ALL DEVICES
IEX$>;
IEX$>; DEMONSTRATION EVENTS ON THE IEC/IEEE BUS
IEX$>;
IEX$>; THE ARGUMENT TO THE 'SEM' COMMAND IS A BIT MAP WHICH
IEX$>; SPECIFIES WHICH EVENTS ON THE IEEE BUS ARE TO BE
IEX$>; RECOGNIZED BY THE DRIVER.  THE 'REV' COMMAND IS THEN
IEX$>; USED TO DETECT THE OCCURRENCE OF THESE EVENTS.
IEX$>;
IEX$>; THE BITS IN THE 'SEM' BIT MAP HAVE THE FOLLOWING MEANING:
IEX$>;
IEX$>;     BIT   CORRESPONDING EVENT
IEX$>;     ===   ===================
IEX$>;      0     SERVICE REQUEST
IEX$>;      1     ADDRESSED AS LISTENER
IEX$>;      2     ADDRESSED AS TALKER
IEX$>;
IEX$>;      3     DEADDRESSED
IEX$>;      4     ADDRESSED AS EXTENDED LISTENER
IEX$>;      5     ADDRESSED AS EXTENDED TALKER
IEX$>;
IEX$>;      6     DEVICE CLEAR
IEX$>;      7     DEVICE TRIGGER
IEX$>;      8     REMOTE/LOCAL CHANGE
IEX$>;
IEX$>;      9     RECEIVE CONTROL
IEX$>;      10    PARALLEL POLL CONFIGURE
IEX$>;      11    PARALLEL POLL UNCONFIGURE
IEX$>;
IEX$>;      12    INTERFACE CLEAR
IEX$>;
IEX$>; IF THE USER ENABLES THE RECOGNITION OF AN EVENT REPRESENTED BY
IEX$>; BITS 2-11, THEN THE BUS WILL BE LOCKED WHEN AN EVENT OCCURS.
IEX$>; THIS INSURES THAT THE USER'S PROGRAM WILL NOT ENCOUNTER TIMING
IEX$>; PROBLEMS IN REACTING TO AN EVENT.  THE BUS MUST BE RELEASED WITH
IEX$>; THE 'DACR' AUXILIARY COMMAND (CODE: 1).
IEX$>;
IEX$>SEM 1 6         ;EVENT ON BEING ADDRESSED AS LISTENER OR TALKER
IEX$>;
IEX$>; ADDRESS IXA1: AS LISTENER
```

EXAMPLE OF IEX EXERCISER EXECUTION USING IEX$DEMO.COM

```
IEX$>;
IEX$>CMD 0 41
IEX$>REV 1           ;RECOGNIZE EVENT
IEX$>WFE 1           ;EXPECTED: ADDRESSED AS LISTENER
    ADDRESSED AS LISTENER
IEX$>AUX 1 1         ;RELEASE HOLDOFF
IEX$>;
IEX$>; SHOW THAT AN EVENT WILL BE GENERATED WHEN A UNIT IS ADDRESSED AS
IEX$>; LISTENER, EVEN IF THE UNIT WAS ALREADY LISTENER.
IEX$>;
IEX$>CMD 0 41        ;ADDRESS DEVICE IXA1: AS LISTENER
IEX$>REV 1           ;RECOGNIZE EVENT
IEX$>WFE 1           ;EXPECTED: ADDRESSED AS LISTENER
    ADDRESSED AS LISTENER
IEX$>AUX 1 1         ;RELEASE HOLDOFF
IEX$>;
IEX$>CMD 0 77        ;UNLISTEN
IEX$>;
IEX$>; NOW PERFORM SIMILAR TESTS ADDRESSING DEVICE IXA1: AS TALKER
IEX$>;
IEX$>;
IEX$>CMD 0 101
IEX$>REV 1           ;RECOGNIZE EVENT
IEX$>WFE 1           ;EXPECTED: ADDRESSED AS TALKER
    ADDRESSED AS TALKER
IEX$>;
IEX$>; NOW DEMONSTRATE THE 'DEADDRESSED' EVENT
IEX$>;
IEX$>SEM 1 12        ;EVENT ON LISTENER OR DEADDRESSED
IEX$>CMD 0 41        ;ADDRESS AS LISTENER
IEX$>REV 1           ;RECOGNIZE EVENT
IEX$>WFE 1           ;EXPECTED: ADDRESSED AS LISTENER
    ADDRESSED AS LISTENER
IEX$>AUX 1 1         ;RELEASE HOLDOFF
IEX$>;
IEX$>CMD 0 77        ;UNTALK
IEX$>REV 1           ;RECOGNIZE EVENT
IEX$>WFE 1           ;EXPECTED: DEADDRESSED
    DEADDRESSED
IEX$>;
IEX$>; DEMONSTRATE USE OF EXTENDED ADDRESSING
IEX$>;
IEX$>SEM 1 60
IEX$>;
IEX$>CMD 0 41 142    ;PRIMARY ADDRESS & SECONDARY ADDRESS
IEX$>REV 1           ;RECOGNIZE EVENT
IEX$>WFE 1           ;EXPECTED: EXTENDED LISTENER
    ADDRESSED AS EXTENDED LISTENER: 142
IEX$>AUX 1 201       ;POSITIVE HOLDOFF RELEASE
IEX$>SNS 1           ;EXPECTED: CIDS, LISTENER
    CONTROLLER IDLE STATE
    ADDRESSED AS LISTENER
    IEEE STATUS REGISTER: 120665
IEX$>CMD 0 77        ;UNLISTEN
IEX$>;
IEX$>CMD 0 101 142   ;PRIMARY ADDRESS & SECONDARY ADDRESS
IEX$>REV 1           ;RECOGNIZE EVENT
IEX$>WFE 1           ;EXPECTED: EXTENDED TALKER
    ADDRESSED AS EXTENDED TALKER: 142
IEX$>AUX 1 201       ;POSITIVE HOLDOFF RELEASE
IEX$>CMD 0 137       ;UNTALK
IEX$>;
IEX$>; DEMONSTRATE THE USE OF DEVICE CLEAR
IEX$>;
IEX$>GCT 0           ;SET IXA0: TO CACS
IEX$>;
IEX$>SEM 1 100       ;EVENT ON DEVICE CLEAR
IEX$>REV 1           ;RECOGNIZE EVENT
IEX$>CMD 0 41 24     ;IXA1: TO LISTENER & DEVICE CLEAR
IEX$>WFE 1           ;EXPECTED: DEVICE CLEAR
```

```
    DEVICE CLEAR
IEX$>AUX 1 1         ;RELEASE HOLDOFF
IEX$>CMD 0 77        ;UNLISTEN
IEX$>;
IEX$>; DEMONSTRATE THAT IF RECOGNITION FOR AN EVENT
IEX$>; IS NOT ENABLED, THEN THE EVENT IS NOT DETECTED
IEX$>; AND THE IEEE-488 BUS IS NOT LOCKED.
IEX$>;
IEX$>SEM 1 0
IEX$>CMD 0 41 24     ;DEVICE CLEAR
IEX$>RES 1           ;TEST FOR EVENT
    NO EVENT SEEN
IEX$>CMD 0 77        ;UNLISTEN
IEX$>;
IEX$>; DEMONSTRATE THE USE OF DEVICE TRIGGER
IEX$>;
IEX$>SEM 1 200       ;EVENT ON DEVICE TRIGGER
IEX$>CMD 0 41 10     ;IXA1: TO LISTEN & GET UNIVERSAL COMMAND
IEX$>REV 1           ;RECOGNIZE EVENT
IEX$>WFE 1           ;EXPECTED: DEVICE TRIGGER
    DEVICE TRIGGER
IEX$>CMD 0 77        ;UNLISTEN
IEX$>;
IEX$>; DEMONSTRATE THE USE OF REMOTE/LOCAL CHANGE
IEX$>;
IEX$>SEM 1 400       ;EVENT ON REMOTE/LOCAL CHANGE
IEX$>AUX 0 20        ;RETURN TO LOCAL
IEX$>CMD 0 41        ;IXA1: TO LISTEN
IEX$>REV 1           ;RECOGNIZE EVENT
IEX$>WFE 1           ;EXPECTED: REMOTE/LOCAL CHANGE
    REMOTE LOCAL CHANGE: 0
IEX$>AUX 0 220       ;RETURN TO REMOTE
IEX$>CMD 0 41 77     ;IXA1: TO LISTEN
IEX$>REV 1           ;RECOGNIZE EVENT
IEX$>WFE 1           ;EXPECTED: REMOTE/LOCAL CHANGE
    REMOTE LOCAL CHANGE: 1
IEX$>SEM 1 0         ;DISABLE ALL EVENT RECOGNITION ON UNIT 1
IEX$>;
IEX$>; DEMONSTRATE THE USE OF PASS CONTROL
IEX$>;
IEX$>SNS 0           ;EXPECTED: CACS, NOT ADDRESSED
    CONTROLLER ACTIVE STATE
    NOT ADDRESSED
    IEEE STATUS REGISTER: 120440
IEX$>SNS 1           ;EXPECTED: CIDS, NOT ADDRESSED
    CONTROLLER IDLE STATE
    NOT ADDRESSED
    IEEE STATUS REGISTER: 120641
IEX$>SEM 1 1000      ;EVENT ON RECEIVE CONTROL
IEX$>PCT 0 101       ;PASS CONTROL TO UNIT 1
IEX$>RES 1           ;EXPECTED: RECEIVED CONTROL
    RECEIVED CONTROL
IEX$>AUX 1 1         ;RELEASE HOLDOFF
IEX$>SNS 0           ;EXPECTED: CIDS, NOT ADDRESSED
    CONTROLLER IDLE STATE
    NOT ADDRESSED
    IEEE STATUS REGISTER: 120440
IEX$>SNS 1           ;EXPECTED: CACS, NOT ADDRESSED
    CONTROLLER ACTIVE STATE
    NOT ADDRESSED
    IEEE STATUS REGISTER: 120641
IEX$>;
IEX$>; NOW GIVE CONTROL BACK TO IXA0:
IEX$>;
IEX$>SEM 0 1000      ;EVENT ON RECEIVE CONTROL
IEX$>PCT 1 100       ;PASS CONTROL TO IXA0:
IEX$>RES 0           ;EXPECTED: RECEIVED CONTROL
    RECEIVED CONTROL
IEX$>AUX 0 1         ;RELEASE HOLDOFF
IEX$>SNS 0           ;EXPECTED: CACS, NOT ADDRESSED
```

```
    CONTROLLER ACTIVE STATE
    NOT ADDRESSED
    IEEE STATUS REGISTER: 120440
IEX$>SNS 1           ;EXPECTED: CIDS, NOT ADDRESSED
    CONTROLLER IDLE STATE
    NOT ADDRESSED
    IEEE STATUS REGISTER: 120641
IEX$>;
IEX$>; DEMONSTRATE THE USE OF ATTENTION ASTS TO RECOGNIZE EVENTS.
IEX$>;
IEX$>; UNSOLICITED ASTS ARE ENABLES THROUGH THE IO$_SETMODE REQUEST
IEX$>; WITH THE IO$_UNS_AST MODIFIER.  WHEN ASTS ARE ENABLED, AN
IEX$>; AST WILL BE QUEUED TO  THE PROCESS THAT ENABLED THEM EVERY
IEX$>; TIME THAT AN ENABLED EVENT OCCURS.  THIS FORM OF RECOGNITION
IEX$>; IS SIMILAR TO RECOGNITION VIA THE IO$_REC_EVENT REQUEST.
IEX$>;
IEX$>SEM 1 6
IEX$>AST 1 ENABLE
IEX$>CMD 0 41
IEX$>WFE 1           ;EXPECTED: ADDRESSED AS LISTENER
    ADDRESSED AS LISTENER
IEX$>SNS 1           ;EXPECTED: CIDS, LISTENER
    CONTROLLER IDLE STATE
    ADDRESSED AS LISTENER
    IEEE STATUS REGISTER: 170665
IEX$>SEM 1 0
IEX$>;
IEX$>;
IEX$>; DEMONSTRATE THE USE OF SERVICE REQUEST AND SERIAL POLL.
IEX$>;
IEX$>; IN A NORMAL SERIAL POLL PROCESS, A DEVICE ON THE IEEE-488
IEX$>; BUS WHICH IS NOT CONTROLLER-IN-CHARGE (C-I-C) ISSUES A SERVICE
IEX$>; REQUEST IN ORDER TO SIGNAL TO THE C-I-C THAT THIS DEVICE
IEX$>; REQUIRES ATTENTION.  THE C-I-C DETECTS THAT THE SERVICE REQUEST
IEX$>; LINE HAS BECOME ASSERTED AND ISSUES A SERIAL POLL
IEX$>; IN ORDER TO DETERMINE WHICH DEVICE OR DEVICES REQUIRE SERVICING.
IEX$>; IN A SERIAL POLL ONE BYTE OF INFORMATION IS RECEIVED FROM EACH
IEX$>; DEVICE ADDRESSED.  A REQUESTING DEVICE CAN CODE THE
IEX$>; TYPE OF SERVICE REQUIRED IN THIS BYTE.
IEX$>;
IEX$>SEM 0 1         ;ENABLE RECOGNITION OF SERVICE REQUEST
IEX$>REV 0           ;RECOGNIZE EVENT
IEX$>RSV 1 134       ;REQUEST SERVICE (BIT 6 PLUS STATUS BYTE)
IEX$>WFE 0           ;EXPECTED: SERVICE REQUEST
    SERVICE REQUEST
IEX$>SPO 0 101       ;PERFORM A SERIAL POLL
NUMBER OF DEVICES POLLED:  1
ENTRY  DEV.ADDR. EXT.ADDR.  STATUS
  1:      101                134
IEX$>;
IEX$>; DEMONSTRATE THE USE OF REQUEST SERVICE AND SERIAL
IEX$>; POLL USING EXTENDED ADDRESSING.
IEX$>;
IEX$>RSV 1 103 142  ;REQUEST SERVICE
IEX$>RES 0
    SERVICE REQUEST
IEX$>SPO 0 101 142  ;SERIAL POLL
NUMBER OF DEVICES POLLED:  1
ENTRY  DEV.ADDR. EXT.ADDR.  STATUS
  1:      101       142      103
IEX$>;
IEX$>; IF A SERVICE REQUEST IS ISSUED WITH BIT 6 CLEARED, THE DRIVER
IEX$>; WILL LOAD THE SERIAL POLL REGISTER BUT NOT MAKE A SERVICE REQUEST.
IEX$>;
IEX$>RSV 1 55
IEX$>RES 0          ;NO EVENT EXPECTED
    NO EVENT SEEN
IEX$>SPO!IO$M_ALLDEVICES 0 101
NUMBER OF DEVICES POLLED:  1
ENTRY  DEV.ADDR. EXT.ADDR.  STATUS
```

```
  1:      101                55
IEX$>;
IEX$>; DEMONSTRATE THE USE OF PARALLEL POLL
IEX$>;
IEX$>; BITS 0-2 OF THE PPC SECONDARY COMMAND INDICATE WITH WHICH
IEX$>; BIT THE ADDRESSED DEVICE WILL RESPOND DURING A PARALLEL
IEX$>; POLL.  BIT 3 SPECIFIES THE STATE OF THAT BIT, IF THE STATUS
IEX$>; OF THE DEVICE FEATURE BEING POLLED IS TRUE.
IEX$>; NOTE: THE DRIVER AUTOMATICALLY SETS BITS 5 AND 6 OF
IEX$>; THE CONFIGURE BYTE.
IEX$>;
IEX$>SEM 1 2000      ;EVENT ON PP CONFIGURE
IEX$>PPC 0 41 3      ;PARALLEL POLL CONFIGURE
IEX$>REV 1           ;RECOGNIZE EVENT
IEX$>WFE 1           ;EXPECTED: PARALLEL POLL CONFIGURE
    PARALLEL POLL CONFIGURE: 143
IEX$>LPP 1 4         ;LOAD PARALLEL POLL REGISTER
IEX$>PPO 0           ;PERFORM A PARALLEL POLL
    PARALLEL POLL STATUS:    4
IEX$>;
IEX$>; DEMONSTRATE HOW A DEVICE IS COMMANDED TO CEASE
IEX$>; PARALLEL POLL OPERATION BY SENDING IT THE PARALLEL
IEX$>; POLL DISABLE (PPD) ADDRESSED COMMAND
IEX$>; NOTE: IF BIT 4 OF THE CONFIGURE BYTE IS SET, THEN
IEX$>; THE COMMAND IS PARALLEL POLL DISABLE (PPD).
IEX$>; NOTE: FOR BOTH PARALLEL POLL ENABLE (PPE) AND PARALLEL
IEX$>; POLL DISABLE (PPD), A PARALLEL POLL CONFIGURE EVENT
IEX$>; IS GENERATED IF RECOGNITION FOR PARALLEL POLL
IEX$>; CONFIGURE WAS ENABLED.
IEX$>;
IEX$>; NOTE: WHEN A PPD COMMAND IS RECEIVED, THE DRIVER
IEX$>; AUTOMATICALLY CLEARS THE PARALLEL POLL REGISTER.
IEX$>;
IEX$>PPC 0 41 160    ;160 IS PPD COMMAND
IEX$>REV 1           ;RECOGNIZE EVENT
IEX$>WFE 1           ;EXPECTED: PARALLEL POLL CONFIGURE
    PARALLEL POLL CONFIGURE: 160
IEX$>AUX 1 1         ;RELEASE BUS
IEX$>PPO 0           ;PERFORM ANOTHER PARALLEL POLL
    PARALLEL POLL STATUS:    0
IEX$>;
IEX$>; A PARALLEL POLL UNCONFIGURE INSTRUCTS ALL DEVICES
IEX$>; TO STOP RESPONDING TO PARALLEL POLLS.  THE DEVICES
IEX$>; DO NOT HAVE TO BE ADDRESSED AS LISTENER TO PROCESS
IEX$>; THIS COMMAND.  THE DRIVER AUTOMATICALLY CLEARS
IEX$>; THE PARALLEL POLL REGISTER WHEN THIS COMMAND IS RECEIVED.
IEX$>;
IEX$>SEM 1 4000      ;EVENT ON PP UNCONFIGURE
IEX$>CMD 0 25        ;ISSUE PARALLEL POLL UNCONFIGURE
IEX$>REV 1           ;RECOGNIZE EVENT
IEX$>WFE 1           ;EXPECTED: PARALLEL POLL UNCONFIGURE
    PARALLEL POLL UNCONFIGURE
IEX$>AUX 1 1         ;RELEASE BUS
IEX$>PPO 0           ;PERFORM ANOTHER PARALLEL POLL
    PARALLEL POLL STATUS:    0
IEX$>;
IEX$>; DEMONSTRATE THE RECOGNITION OF INTERFACE CLEAR EVENTS
IEX$>; AN INTERFACE CLEAR EVENT IS OF PARTICULAR INTEREST TO A C-I-C
IEX$>; WHICH IS NOT SYSTEM CONTROLLER.  SUCH AN EVENT INDICATES
IEX$>; THAT THE SYSTEM CONTROLLER HAS BECOME C-I-C.  ANY UNIT
IEX$>; WHICH RECOGNIZES A INTERFACE CLEAR EVENT WILL AUTOMATICALLY
IEX$>; BE SET TO CIDS BY THE DRIVER.
IEX$>;
IEX$>SEM 1 10000     ;EVENT ON INTERFACE CLEAR
IEX$>REV 1           ;LOOK FOR EVENT
IEX$>AUX 0 217       ;SEND INTERFACE CLEAR
IEX$>AUX 0 17        ;RESET IFC LINE
IEX$>WFE 1           ;EXPECTED EVENT: INTERFACE CLEAR
    INTERFACE CLEAR
IEX$>SNS 0           ;SENDING INTERFACE CLEAR SETS C-I-C TO CACS
```

EXAMPLE OF IEX EXERCISER EXECUTION USING IEX$DEMO.COM

```
    CONTROLLER ACTIVE STATE
    NOT ADDRESSED
    IEEE STATUS REGISTER: 120640
IEX$>;
IEX$>;
IEX$>; DEMONSTRATE DATA TRANSFERS
IEX$>;
IEX$>; DEMONSTRATE THE WAYS IN WHICH A READ MAY TERMINATE.
IEX$>;
IEX$>; FIRST DEMONSTRATE THE USE OF BYTE COUNT OVERFLOW.
IEX$>;
IEX$>ENABLE CHK      ;COMPARE DATA SENT WITH DATA RECEIVED
IEX$>AUX 0 212       ;0 TO TALKER
IEX$>CMD 0 41        ;1 TO LISTENER
IEX$>RLB 1 50.       ;1 READS 50 BYTES
IEX$>WLB 0 50.       ;THIS SHOULD RESULT IN READ REQUEST TERMINATING
IEX$>WFT 1           ;WAIT FOR TRANSFER (DOES NOT GENERATE QIO TO DRIVER)
  Read Complete:     50. bytes,     Termination Method:   BYTE COUNT MET
IEX$>WFT 0           ;WAIT FOR TRANSFER (DOES NOT GENERATE QIO TO DRIVER)
  Write Complete:    50.  bytes
IEX$>;
IEX$>; PRINT DATA FROM LAST READ REQUEST
IEX$>;
IEX$>PRINT
    1.:       0      1      2      3      4      5      6      7     10     11
   11.:      12     13     14     15     16     17     20     21     22     23
   21.:      24     25     26     27     30     31     32     33     34     35
   31.:      36     37     40     41     42     43     44     45     46     47
   41.:      50     51     52     53     54     55     56     57     60     61
IEX$>;
IEX$>; NOW DEMONSTRATE THE USE OF IO$M_EOI.  WITH THIS METHOD OF TRANSFER
IEX$>; THE EOI LINE IS ASSERTED DURING THE TRANSMISSION OF THE LAST BYTE.
IEX$>;
IEX$>RLB 1 100.
IEX$>WLB!IO$M_EOI 0 40.  ;READ REQUEST SHOULD TERMINATE AFTER 40 BYTES
IEX$>WFT 1               ;WAIT FOR TRANSFER (DOES NOT GENERATE QIO TO DRIVER)
  Read Complete:     40. bytes,     Termination Method:   EOI DETECTED
IEX$>WFT 0               ;SHOW THAT WRITE ALSO COMPLETES
  Write Complete:    40.  bytes
IEX$>;
IEX$>; NOW DEMONSTRATE THE USE OF MATCH CHARACTER TERMINATION
IEX$>; IN THIS MODE THE RECEPTION OF DATA TERMINATES AFTER THE
IEX$>; MATCH CHARACTER HAS BEEN CONSECUTIVELY DETECTED THE SPECIFIED
IEX$>; NUMBER OF TIMES.
IEX$>;
IEX$>RLB 1 400. 100 1 ;(MATCH CHAR = '100', MATCH COUNT = 1)
IEX$>WLB 0 101
IEX$>WFT 1           ;WAIT FOR TRANSFER (DOES NOT GENERATE QIO TO DRIVER)
  Read Complete:     65. bytes,     Termination Method:   MATCH CHARACTER
IEX$>WFT 0           ;SHOW THAT WRITE ALSO COMPLETES
  Write Complete:    65.  bytes
IEX$>;
IEX$>; DEMONSTRATE THE USE OF THE QIO P5 LISTENER PARAMETER FOR THE WRITE FUNCTION
IEX$>;   NOTE:  ONLY THE C-I-C CAN ISSUE A WRITE REQUEST WITH THE P5 LISTENER ADDRESS
IEX$>;
IEX$>CMD 0 77 137    ;UNTALK AND UNLISTEN ALL
IEX$>CMD 0 100       ;UNIT 0 AS TALKER
IEX$>WLB 0 60. 41    ;SEND MLA TO DEVICE 1 AND WRITE 60 BYTES
IEX$>RLB 1 60.       ;READ ON DEVICE 1 DATA SENT
IEX$>WFT 0           ;WAIT FOR TRANSFER (DOES NOT GENERATE QIO TO DRIVER)
  Write Complete:    60.  bytes
IEX$>WFT 1           ;SHOW THAT READ ALSO COMPLETES
  Read Complete:     60. bytes,     Termination Method:   BYTE COUNT MET
IEX$>;
IEX$>; DEMONSTRATE THE USE OF THE QIO P5 TALKER PARAMETER FOR THE READ FUNCTION
IEX$>;   NOTE:  ONLY THE C-I-C CAN ISSUE A READ REQUEST WITH THE P5 TALKER ADDRESS
IEX$>;
IEX$>CMD 0 77 137    ;UNTALK AND UNLISTEN ALL
IEX$>CMD 0 40        ;UNIT 0 AS LISTENER
IEX$>RLB 0 70. 101   ;SEND MTA TO DEVICE 1 AND READY TO READ DATA
```

```
IEX$>WLB 1 70.        ;WRITE 70 BYTES
IEX$>WFT 0            ;WAIT FOR TRANSFER (DOES NOT GENERATE QIO TO DRIVER)
  Read Complete:      70. bytes,     Termination Method:   BYTE COUNT MET
IEX$>WFT 1            ;SHOW THAT READ ALSO COMPLETES
  Write Complete:     70.  bytes
IEX$>;
IEX$>; SHOW THAT IT DOESN'T MATTER WHICH REQUEST COMES FIRST
IEX$>;
IEX$>CMD 0 77 137     ;UNTALK AND UNLISTEN ALL
IEX$>CMD 0 100        ;ADDRESS AS TALKER
IEX$>CMD 0 41         ;ADDRESS AS LISTENER
IEX$>WLB 0 500.
IEX$>RLB 1 500.
IEX$>WFT 0            ;WAIT FOR TRANSFER (DOES NOT GENERATE QIO TO DRIVER)
  Write Complete:    500.  bytes
IEX$>WFT 1            ;SHOW THAT READ ALSO COMPLETES
  Read Complete:     500. bytes,     Termination Method:   BYTE COUNT MET
IEX$>;
IEX$>; SHOW THE AN IO$_SENSEMODE REQUEST WITHOUT MODIFIERS IS PROCESSED
IEX$>; WITHOUT BEING QUEUED.
IEX$>;
IEX$>RLB 1 100.        ;THIS REQUEST WILL HANG
IEX$>SNS 1            ;EXPECTED: CIDL, LISTENER
    CONTROLLER IDLE STATE
    ADDRESSED AS LISTENER
    IEEE STATUS REGISTER: 120665
IEX$>CAN 1            ;CANCEL RLB REQUEST
IEX$>;
IEX$>; USE SECONDARY ADDRESSING TO ADDRESS A DEVICE AND THEN SEND IT DATA
IEX$>;
IEX$>SEM 1 20         ;RECOGNIZE EXTENDED ADDRESSING
IEX$>CMD 0 41 142     ;PRIMARY ADDRESS _& SECONDARY ADDRESS
IEX$>RES 1            ;RECOGNIZE EVENT
    ADDRESSED AS LISTENER
IEX$>WFE 1            ;EXPECTED: EXTENDED LISTENER
    ADDRESSED AS EXTENDED LISTENER: 142
IEX$>AUX 1 201        ;POSITIVE HOLDOFF RELEASE
IEX$>SNS 1            ;EXPECTED: CIDS, LISTENER
    CONTROLLER IDLE STATE
    ADDRESSED AS LISTENER
    IEEE STATUS REGISTER: 120665
IEX$>AUX 0 212        ;UNIT 0 IS TALKER
IEX$>RLB 1 100.
IEX$>WLB 0 100.
IEX$>WFT 0            ;WAIT FOR TRANSFER
  Write Complete:    100.  bytes
IEX$>WFT 1            ;WAIT FOR TRANSFER
  Read Complete:     100. bytes,     Termination Method:   BYTE COUNT MET
IEX$>SEM 1 0          ;DISABLE EXTENDED MODE ADDRESSING
IEX$>;
IEX$>; IF THE IEX11 IS USED TO TRANSFER DATA WITH OTHER DEVICES
IEX$>; WHICH ARE CAPABLE OF FAST TRANSFERS, THE INTERNAL TIMING
IEX$>; OF THE IEX11 CAN BE MADE FASTER.  AN EXAMPLE OF FAST
IEX$>; DATA TRANSFERS IS A COMPUTER TO COMPUTER CONNECTION
IEX$>; USING THE IEEE-488 BUS.
IEX$>;
IEX$>AUX 0 225        ;SET T1 DELAY TO 6 CLOCK CYCLES
IEX$>AUX 0 212        ;IXA0: TO TALKER
IEX$>CMD 0 41         ;IXA1: TO LISTENER
IEX$>RLB 1 500.       ;500. BYTES
IEX$>WLB 0 500.
IEX$>WFT 0            ;WAIT FOR TRANSFER
  Write Complete:    500.  bytes
IEX$>WFT 1            ;WAIT FOR TRANSFER
  Read Complete:     500. bytes,     Termination Method:   BYTE COUNT MET
IEX$>AUX 0 227        ;SET T1 TO VERY SHORT DELAY TIME, 3 CLOCK CYCLES
IEX$>RLB 1 500.       ;500. BYTES
IEX$>WLB 0 500.
IEX$>WFT 0            ;WAIT FOR TRANSFER
  Write Complete:    500.  bytes
```

EXAMPLE OF IEX EXERCISER EXECUTION USING IEX$DEMO.COM

```
IEX$>WFT 1           ;WAIT FOR TRANSFER
  Read Complete:    500. bytes,     Termination Method:   BYTE COUNT MET
IEX$>AUX 0 27        ;RESET DELAY TIMES
IEX$>AUX 0 25        ;MUST BE DONE ALSO
IEX$>;
IEX$>; DEMONSTRATE THE USE OF THE 'SET DATA' COMMAND
IEX$>;
IEX$>SET DATA C 0 SEND THIS MESSAGE
IEX$>AUX 1 211       ;#1 TO LISTENER
IEX$>RLB 1 100.
IEX$>WLB!IO$M_EOI 0 17.  ;SEND MESSAGE
IEX$>WFT 1               ;WAIT FOR TRANSFER (DOES NOT GENERATE QIO TO DRIVER)
  Read Complete:     17. bytes,     Termination Method:   EOI DETECTED
IEX$>WFT 0               ;SHOW THAT WRITE ALSO COMPLETES
  Write Complete:    17.  bytes
IEX$>PRINT C             ;PRINT DATA IN CHARACTER MODE
SEND THIS MESSAGE
IEX$>;
IEX$>; SIMULATE A LINE PROTOCOL
IEX$>;
IEX$>SET DATA C 0 <1>MESSAGE<3>
IEX$>VERIFY          ;THIS COMMAND PRINTS THE WRITE BUFFER
<1>MESSAGE<3>
IEX$>RLB 1 100.
IEX$>WET 0 9.        ;WRITE MESSAGE WITH EOI TERMINATION
IEX$>WFT 1           ;WAIT FOR TRANSFER (DOES NOT GENERATE QIO TO DRIVER)
  Read Complete:      9. bytes,     Termination Method:   EOI DETECTED
IEX$>WFT 0           ;SHOW THAT WRITE ALSO COMPLETES
  Write Complete:     9.  bytes
IEX$>PRINT C
<1>MESSAGE<3>
IEX$>;
IEX$>; SEND SOME HEXADECIMAL DATA
IEX$>;
IEX$>SET DATA X 0 FF FE FD FC FB FA F0 0
IEX$>SET DATA X 8.  2F 3F 4F 5F 6F 7F 8F 9F
IEX$>VERIFY          ;THIS COMMAND PRINTS THE WRITE BUFFER
    1.:      FF     FE     FD     FC     FB     FA     F0      0     2F     3F
   11.:      4F     5F     6F     7F     8F     9F
IEX$>RLB 1 100.
IEX$>WLB!IO$M_EOI 0 16.
IEX$>WFT 1           ;WAIT FOR TRANSFER (DOES NOT GENERATE QIO TO DRIVER)
  Read Complete:     16. bytes,     Termination Method:   EOI DETECTED
IEX$>WFT 0           ;SHOW THAT WRITE ALSO COMPLETES
  Write Complete:    16.  bytes
IEX$>PRINT X
    1.:      FF     FE     FD     FC     FB     FA     F0      0     2F     3F
   11.:      4F     5F     6F     7F     8F     9F
IEX$>PRINT
    1.:     377    376    375    374    373    372    360      0     57     77
   11.:     117    137    157    177    217    237
IEX$>PRINT D         ;DECIMAL FORMAT
    1.:     255    254    253    252    251    250    240      0     47     63
   11.:      79     95    111    127    143    159
IEX$>PRINT W
    1.:  177377 176375 175373    360  37457  57517  77557 117617
IEX$>PRINT WD
    1.:    -257   -771  -1285    240  16175  24399  32623 -24689
IEX$>PRINT WX
    1.:    FEFF   FCFD   FAFB     F0   3F2F   5F4F   7F6F   9F8F
IEX$>;
IEX$>; FOR SIMPLE STRINGS THE USER MAY USE THE FOLLOWING FORM
IEX$>;
IEX$>DISABLE CHK     ;DISABLE DATA CHECKING
IEX$>RLB 1 100
IEX$>WLB 0 <1>F100.12R<3>
IEX$>WFT 0
  Write Complete:    10.  bytes
IEX$>WFT 1
  Read Complete:     10. bytes,     Termination Method:   EOI DETECTED
```

```
IEX$>PRINT C
<1>F100.12R<3>
IEX$>CMD 0 77 137    ;UNTALK AND UNLISTEN ALL
IEX$>AUX 0 217       ;SEND INTERFACE CLEAR
IEX$>AUX 0 17        ;RESET IFC LINE
IEX$>SEM 0 0         ;DISABLE EVENTS
IEX$>SEM 1 0         ;DISABLE EVENTS
IEX$>;
IEX$>;
IEX$>; END OF DEMONSTRATION OF DATA TRANSFER SECTION
IEX$>;
IEX$>;-------------------------------------------------------------------------;
IEX$>;        IF NO ERROR MESSAGES HAVE BEEN PRODUCED, THEN THE IEx11          ;
IEX$>;        INTERFACE AND IEX-VMS DRIVER ARE FUNCTIONING PROPERLY.           ;
IEX$>;-------------------------------------------------------------------------;
IEX$>EXIT
$
```

B EXAMPLE OF PROGRAM CONVERSION FROM IEC11 TO IEX11

The following program is distributed with the IEC11-V driver. It demonstrates data transfers between an IEC11-A and an IEC11-B unit.

The IEC11 version of the program will generate 'state change to LACS' and 'state change to TACS' events. The IEX-VMS-Driver will only generate an *addressed as talker* event, because the unit is addressed as listener after the read data request has been issued.

B.1 ORIGINAL IEC11 PROGRAM

```
        .TITLE   IEABMDEM - IEC11-A TO IEC11-B TRANSFER DEMONSTRATION
        .IDENT   /IEC11-V V3.0-1.4/

; COPYRIGHT (C) 1980 DIGITAL EQUIPMENT CORP.,
; COMPUTER SPECIAL SYSTEMS
; NASHUA, N.H. 03060
;
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A
; SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH THE INCLUSION
; OF THE ABOVE COPYRIGHT NOTICE.  THIS SOFTWARE,  OR ANY OTHER
; COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE
; TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH SYSTEM AND TO ONE WHO
; AGREES TO THESE LICENSE TERMS.  TITLE TO AND OWNERSHIP OF THE
; SOFTWARE SHALL AT ALL TIMES REMAIN IN DEC.
;
; THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT
; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
; EQUIPMENT CORPORATION.
;
; DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
; ITS SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.
;
;++
; FACILITY:  IEEE-488 BUS
;
; FUNCTIONAL DESCRIPTION:
;
;       THIS PROGRAM CONTAINS A SAMPLE INTERACTION BETWEEN AN
;       IEC11-A AND AN IEC11-B.  FIRST, THE IEC11-B IS ADDRESSED
;       AS A LISTENER AND THE IEC11-A AS A TALKER AND A CHARACTER
;       STRING IS TRANSFERRED.  THEN THE TALKER/LISTENER ROLES ARE
;       REVERSED AND THE STRING IS TRANSFERRED IN THE OPPOSITE
;       DIRECTION.  THE MINIMUM NUMBER OF QIO'S NECESSARY TO ACCOMPLISH
;       THIS TRANSACTION (NO POLLING OR AST CAPABILITY) HAVE THE ; *A
;       FLAG IN THE COMMENT FOR THE IEC11-A CONTROLLING PROCESS AND THE
;       ; *B FOR THE IEC11-B PROCESS
;
;       BOTH IEC11-B TRANSFER FUNCTIONS REQUEST SERVICE AND THE
;       IEC11-A THEN PROCEEDS TO CONDUCT A SERIAL POLL.  THE RESULTS
;       OF THIS SERIAL POLL SHOULD BE A STATUS BYTE WITH OCTAL
;       VALUE 104 INDICATING THAT THE IEC11-B REQUESTED SERVICE
;       WITH THE OVERFLOW FLAG SET.
;
;       SINCE THE IEC11-B IS ADDRESSED AS TALKER BEFORE THIS
;       CONTROLLER EXPECTS TO TALK, THERE SHOULD BE AN UNSOLICITED
;       INTERRUPT WITH TACS (TALKER ACTIVE) FLAGGED IN THE PARAMETER
;
;       NOTE: THIS PROGRAM ASSUMES THAT AN IEC11-A HAS BEEN
;       ASSIGNED THE DEVICE MNEUMONIC IEA0: BY SYSGEN AND THAT
```

```
;       AN IEC11-B HAS BEEN ASSIGNED THE DEVICE MNEUMONIC IEB0:
;
;       ALTHOUGH THE $CLREF (CLEAR EVENT FLAG) STATEMENTS HAVE NO
;       PURPOSE IN THIS PROGRAM, THEY ARE INCLUDED TO ILLUSTRATE
;       THE USE OF THE EVENT FLAG IN THE IO$_SETMODE!IO$M_UNS_AST CALL
;
; ENVIRONMENT:  VAX/VMS MACRO32
;
; AUTHOR:  Edward J. Los,   CREATION DATE:  06-Oct-80
;
; MODIFIED BY:
;
;
;       E. J. Los, 19-Feb-82: PATCHED 1.3                                      ;EJL001
; Add general references (G^) for the V3.0 compiler                           ;EJL001
;
;       E. J. Los, 18-Apr-84: PATCHED 1.4
; Correct IEC11-B initialization command.
;                                                                              ;EJL001
;--
        .PAGE
        .SBTTL  IEABMDEM - IEC11-A TO IEC11-B TRANSFER DEMONSTRATION
        .PSECT  .DATA,QUAD
        .ENABL  GBL
        $IECDEF                         ; IEC CODE DEFINITIONS
;
;        QIO BUFFERS
;
IOA_STATUS:     .BLKQ   1               ; I/O STATUS BLOCK
IOB_STATUS:     .BLKQ   1               ; I/O STATUS BLOCK FOR IEC11-B
IOP_STATUS:     .LONG   1,0             ; SERIAL POLL STATUS
;
;     MISCELLANEOUS STORAGE LOCATIONS
;
IEACHAN:        .BLKL   1               ; IEC11-A CHANNEL NUMBER
IBACHAN:        .BLKL   1               ; IEC11-B
ERRLOC:         .BLKL   1               ; ERROR RETURN LOCATION
A_ADDRESS:      .BLKL   1               ; IEC11-A BUS ADDRESS
B_ADDRESS:      .BLKL   1               ; IEC11-B BUS ADDRESS
BUFLEN:         .BLKL   1               ; LENGTH OF INPUT BUFFER
RESBUF:         .BLKL   1               ; SERIAL POLL RESPONSE BUFFER
;
;       MESSAGE TEXTS
;
                .MACRO  DESCRIPTOR      TEXT,?LABEL1,?LABEL2
                .LONG   LABEL2-LABEL1
                .LONG   LABEL1
LABEL1:         .ASCII  /TEXT/
LABEL2:
                .ENDM   DESCRIPTOR
                .MACRO  OUTPUT_ARG      TEXT,?LABEL3,?LABEL4
LABEL4:         .LONG   1
                .LONG   LABEL3

LABEL3:         DESCRIPTOR      <TEXT>
                .ENDM   OUTPUT_ARG
IEANAMDSC:      DESCRIPTOR      <IEA0>  ; IEC11-A NAME DESCRIPTOR
IBANAMDSC:      DESCRIPTOR      <IBA0>  ; IEC11-B NAME DESCRIPTOR
APRDSC:         DESCRIPTOR      <WHAT IS THE IEC11-A IEEE BUS ADDRESS (DEC)? >
BPRDSC:         DESCRIPTOR      <WHAT IS THE IEC11-B IEEE BUS ADDRESS (DEC)? >
STRTMSG:
OUTPUT_ARG <IEC11-A TO IEC11-B MACRO TRANSFER DEMONSTRATION - IEC11-V V3.0>
ABMSG:          OUTPUT_ARG      <TRANSFER FROM THE IEC11-A TO THE IEC11-B>
BAMSG:          OUTPUT_ARG      <TRANSFER FROM THE IEC11-B TO THE IEC11-A>
ENDMSG:         OUTPUT_ARG      <END OF TRANSFER DEMONSTRATION>
ERRMSG:         OUTPUT_ARG      <ERROR DETECTED BEFORE PC:>
LOCMSG:         OUTPUT_ARG      <12345678>,LOCDSC
POLL_ERR_MSG:   OUTPUT_ARG      <ERROR WHILE CONDUCTING SERIAL POLL>
AST_UNR_MSG:    OUTPUT_ARG      <UNRECOGNIZED IEC11-A UNSOLICITED INTERRUPT>
AST_ERR_MSG:    OUTPUT_ARG      <IEC11-A HARDWARE ERROR INTERRUPT>
```

```
RESMSG:         .LONG   1               ; RESULT TEXT ARGUMENT LIST
                .LONG   RESDSC
RESDSC:         .LONG   RESTXTSIZ       ; RESULT MESSAGE DESCRIPTOR
                .LONG   RESTXT
RSPDSC:         .LONG   4               ; SERIAL POLL RESPONSE DESCRIPTOR
                .LONG   RSPTXT
BUFDSC:         .LONG   BUFTXTSIZ       ; CONVERSION BUFFER DESCRIPTOR
                .LONG   BUFFER
POLLMSG:        .LONG   1               ; SERIAL POLL MESSAGE ARG. LIST
                .LONG   POLLDSC
POLLDSC:        .LONG   POLLTXTSIZ      ; SERIAL POLL MESSAGE DESCRIPTOR
                .LONG   POLLTXT
AST_STATE_MSG:  .LONG   1               ; UNSOLICITED STATE CHANGE ARGUMENTS
                .LONG   ASTSTDSC
ASTSTDSC:       .LONG   ASTSTTXTSIZ
                .LONG   ASTSTTXT
RESTXT:         .ASCII  / TRANSFER BUFFER: /
TRNBUF:         .ASCII  /................................/
RESTXTSIZ       =.-RESTXT
FORTXT:         .ASCII  /ABCDEFGHIJKLMNOPQRSTUVWXYZ/
FORTXTSIZ       =.-FORTXT
REVTXT:         .ASCII  /ZYXWVUTSRQPONMLKJIHGFEDCBA/
REVTXTSIZ       =.-REVTXT
BUFFER:         .ASCII  /0000/
BUFTXTSIZ       =.-BUFFER
ASTSTTXT:       .ASCII  / UNSOLICITED IEC11-/
AST_DEV_MSG:    .BLKB   1
                 .ASCII / STATE CHANGE TO: /
STATE_TXT:      .BLKB   4
ASTSTTXTSIZ     =.-ASTSTTXT
POLLTXT:        .ASCII  / SERIAL POLL RESPONSE BYTE FROM IEC11-B IS (HEX): /
RSPTXT:         .BLKB   4
POLLTXTSIZ      =.-POLLTXT
DEV_CHAR:       .ASCII  /AB/            ; AST DEVICE RECOGNITION
STATES:         .ASCII  /XXXXTACSLACSSPASCACSCIDSCSBS/ ; LIST OF STATES
CMDBUF:         .BYTE   ^O137,^O77      ; UNTALK, UNLISTEN
LTNBYT:         .BLKB   1               ; LISTENER COMMAND
TLKBYT:         .BLKB   1               ; TALKER COMMAND
CMDLEN          =.-CMDBUF
DEVBUF:         .BLKB   1               ; SERIAL POLL DEVICE LIST
        .PAGE
        .PSECT  CODE,NOWRT
;
;       PROGRAM ENTRY POINT
;
        .ENABL  LSB
        .ENTRY  IEABMDEM,0
        CALLG   STRTMSG,G^LIB$PUT_OUTPUT ; WRITE STARTING MESSAGE              ;EJL001
        BSBW    TST_ERROR                ; CHECK FOR ERROR                      ;**-1

;
;...    <Request the IEEE bus addresses of the IEC11-A and the IEC11-B>
;
```

```
        PUSHAL  APRDSC                  ; REQUEST IEC11-A ADDRESS
        PUSHAL  BUFDSC
        CALLS   #2,G^LIB$GET_INPUT                                              ;EJL001
        BSBW    TST_ERROR               ; CHECK FOR ERROR                       ;**-1
        LOCC    #^A/ /,#BUFTXTSIZ,BUFFER ; FIND END OF INPUT STRING
        SUBL3   R0,#BUFTXTSIZ,BUFLEN    ; COMPUTE LENGTH OF STRING
        PUSHAL  A_ADDRESS               ; CONVERT ADDRESS FROM DECIMAL
        PUSHAL  BUFFER
        PUSHL   BUFLEN
        CALLS   #3,G^LIB$CVT_DTB                                                ;EJL001
        BSBW    TST_ERROR               ; CHECK FOR ERROR                       ;**-1
        PUSHAL  BPRDSC                  ; REQUEST IEC11-B ADDRESS
        PUSHAL  BUFDSC
        CALLS   #2,G^LIB$GET_INPUT                                              ;EJL001
        BSBW    TST_ERROR               ; CHECK FOR ERROR                       ;**-1
        LOCC    #^A/ /,#BUFTXTSIZ,BUFFER ; FIND END OF INPUT STRING
        SUBL3   R0,#BUFTXTSIZ,BUFLEN    ; COMPUTE LENGTH OF STRING
        PUSHAL  B_ADDRESS               ; CONVERT ADDRESS FROM DECIMAL
        PUSHAL  BUFFER
        PUSHL   BUFLEN
        CALLS   #3,G^LIB$CVT_DTB                                                ;EJL001
        BSBW    TST_ERROR               ; CHECK FOR ERROR                       ;**-1

;
;...    <Assign channels>
;

        $ASSIGN_S DEVNAM=IEANAMDSC,-    ; ASSIGN A CHANNEL FOR THE IEC11-A
                  CHAN=IEACHAN
        BSBW    TST_ERROR               ; CHECK FOR ERROR
        $ASSIGN_S DEVNAM=IBANAMDSC,-    ; ASSIGN A CHANNEL FOR THE IEC11-B
                  CHAN=IBACHAN
        BSBW    TST_ERROR               ; CHECK FOR ERROR

;
;...    <Initialize the IEC11-A, and enable unsolicited interrupts>
;

        $QIOW_S CHAN = IEACHAN,-        ; *A INITIALIZE WITH CACS (CONTROLLER
                FUNC = #IO$_INITIALIZE!IO$M_CACS,- ;  ACTIVE) STATE
                EFN  = #1,-
                IOSB = IOA_STATUS
        BSBW    STSA_ERROR              ; CHECK FOR ERROR
        $QIOW_S CHAN = IEACHAN,-        ; ENABLE UNSOLICITED INTERRUPT ASTS
                FUNC = #IO$_SETMODE!IO$M_UNS_AST,- ;
                EFN  = #2,-
                IOSB = IOA_STATUS,-     ;
                ASTADR = INAAST         ;
        BSBW    STSA_ERROR              ; CHECK FOR ERROR
        $CLREF_S EFN = #2               ; CLEAR THE EVENT FLAG
        BSBW    TST_ERROR               ; CHECK FOR ERROR
        ADDB3   #^O100,B_ADDRESS,DEVBUF ; PUT IEC11-B MTA ADDRESS IN SERIAL
                                        ;  POLL LIST
        $QIOW_S CHAN = IEACHAN,-        ; ENABLE SERVICE REQUEST INTERRUPTS
                FUNC = #IO$_SETMODE!IO$M_ENA_SRQ,-
                EFN  = #1,-
                IOSB = IOA_STATUS
        BSBW    STSA_ERROR              ; CHECK FOR ERROR

;
;...    <Initialize the IEC11-B, and enable unsolicited interupts>
;
```

```
        $QIOW_S CHAN = IBACHAN,-         ; *B INITIALIZE IEC11-B
                FUNC = #IO$_INITIALIZE,- ;
                EFN  = #1,-
                IOSB = IOB_STATUS
        BSBW    STSB_ERROR               ; CHECK FOR ERROR
        $QIOW_S CHAN = IBACHAN,-         ; ENABLE UNSOLICITED INTERRUPT ASTS
                FUNC = #IO$_SETMODE!IO$M_UNS_AST,-
                EFN  = #3,-
                IOSB = IOB_STATUS,-      ;
                ASTADR = INBAST          ;
        BSBW    STSB_ERROR               ; CHECK FOR ERROR
        $CLREF_S EFN = #3                ; CLEAR THE EVENT FLAG
        BSBW    TST_ERROR                ; CHECK FOR ERROR

;
;...    <Transfer a string from the IEC11-A to the IEC11-B>
;

        CALLG   ABMSG,G^LIB$PUT_OUTPUT  ; WRITE A-B TRANSFER MESSAGE           ;EJL001
        BSBW    TST_ERROR                ; CHECK FOR ERROR                      ;**-1
        $QIO_S  CHAN = IBACHAN,-         ; *B IEC11-B READ WHEN ADDRESSED
                FUNC = #IO$_READVBLK!IO$M_REQSRV,- ; AS LACS AND REQUEST
                IOSB = IOB_STATUS,-      ;  SERVICE AFTERWARDS
                EFN  = #4,-              ;
                P1 = TRNBUF,-            ; TRANSFER BUFFER
                P2 = #FORTXTSIZ          ; EXPECTED BUFFER LENGTH
        BSBW    TST_ERROR                ; CHECK FOR ERROR
        ADDB3   #^O40,B_ADDRESS,LTNBYT  ; FIND MY LISTEN ADDRESS FOR IEC11-B
        ADDB3   #^O100,A_ADDRESS,TLKBYT ; FIND MY TALK ADDRESS FOR IEC11-A
        $QIOW_S CHAN = IEACHAN,-         ; *A SEND COMMAND TO ADDRESS THE
                FUNC = #IO$_WRITEVBLK!IO$M_COMMAND,- ; IEC11-A AS TALKER AND
                EFN  = #1,-              ; THE IEC11-B AS LISTENER
                IOSB = IOA_STATUS,-      ;
                P1 = CMDBUF,-            ; COMMAND BUFFER
                P2 = #CMDLEN             ; COMMAND BUFFER LENGTH
        BSBW    STSA_ERROR               ; CHECK FOR ERROR
        $QIOW_S CHAN = IEACHAN,-         ; *A SEND BUFFER WITHOUT AN EOI
                FUNC = #IO$_WRITEVBLK,- ;    OR MATCH CHARACTERS
                EFN  = #1,-              ;
                IOSB = IOA_STATUS,-      ;
                P1 = FORTXT,-            ; FORWARD TRANSFER BUFFER
                P2 = #FORTXTSIZ          ; BUFFER LENGTH
        BSBW    STSA_ERROR               ; CHECK FOR ERROR
        $WAITFR_S       EFN = #4         ; WAIT FOR IEC11-B COMPLETION
        BSBW    STSB_ERROR               ; CHECK FOR ERROR
        CALLG   RESMSG,G^LIB$PUT_OUTPUT ; WRITE THE RESULTS                    ;EJL001
;                                                                               ;**-1
;...    <Transfer a string from the IEC11-B to the IEC11-A>
;
```

```
        CALLG   BAMSG,G^LIB$PUT_OUTPUT  ; WRITE B-A TRANSFER MESSAGE            ;EJL001
        BSBW    TST_ERROR               ; CHECK FOR ERROR                       ;**-1
        ADDB3   #^O40,A_ADDRESS,LTNBYT  ; FIND MY LISTEN ADDRESS FOR IEC11-A
        ADDB3   #^O100,B_ADDRESS,TLKBYT ; FIND MY TALK ADDRESS FOR IEC11-B
        $QIOW_S CHAN = IEACHAN,-        ; *A SEND COMMAND TO ADDRESS THE
                FUNC = #IO$_WRITEVBLK!IO$M_COMMAND,- ; IEC11-A AS LISTENER
                EFN  = #1,-             ; AND THE IEC11-B AS TALKER
                IOSB = IOA_STATUS,-     ;
                P1 = CMDBUF,-           ; COMMAND BUFFER
                P2 = #CMDLEN            ; COMMAND BUFFER LENGTH
        BSBW    STSA_ERROR              ; CHECK FOR ERROR
        $QIO_S  CHAN = IEACHAN,-        ; *A RECEIVE BUFFER, END WITH
                FUNC = #IO$_READVBLK,-  ;   BYTE COUNT SATISFIED OR EOI
                EFN  = #4,-
                IOSB = IOA_STATUS,-     ;
                P1 = TRNBUF,-           ; TRANSFER BUFFER
                P2 = #REVTXTSIZ         ; EXPECTED BUFFER LENGTH
        BSBW    TST_ERROR               ; CHECK FOR ERROR
        $QIOW_S CHAN = IBACHAN,-        ; *B IEC11-B WRITE THE BUFFER AND
                FUNC = #IO$_WRITEVBLK!IO$M_REQSRV,- ; REQUEST SERVICE WHEN
                IOSB = IOB_STATUS,-     ;    DONE
                EFN  = #1,-             ;
                P1 = REVTXT,-           ; REVERSE TRANSFER BUFFER
                P2 = #REVTXTSIZ         ; BUFFER LENGTH
        BSBW    STSB_ERROR              ; CHECK FOR ERROR
        $WAITFR_S       EFN = #4        ; WAIT FOR IEC11-B COMPLETION
        BSBW    STSA_ERROR              ; CHECK FOR ERROR
        CALLG   RESMSG,G^LIB$PUT_OUTPUT ; WRITE THE RESULTS                     ;EJL001
;                                                                               ;**-1
;...    <EXIT>
;
10$:    CALLG   ENDMSG,G^LIB$PUT_OUTPUT ; WRITE END MESSAGE                     ;EJL001
        BSBW    TST_ERROR               ; CHECK FOR ERROR                       ;**-1
        BRW     FINISH
;
;       STATUS CHECKING ROUTINES
;
;
;       CHECK IEC11-A STATUS, INCLUDING SERIAL POLL STATUS FROM
;       THE AST ROUTINE
;
STSA_ERROR:
        BLBC    R0,ERROR                ; BRANCH ON DIRECTIVE ERROR
        MOVL    IOA_STATUS,R0           ; OBTAIN I/O STATUS
        BLBC    R0,ERROR                ; BRANCH ON I/O STATUS ERROR
        MOVL    IOP_STATUS,R0           ; GET STATUS FROM LAST AST
        BLBS    R0,20$                  ; BRANCH IF OK
        TSTL    (SP)+                   ; GET RID OF RETURN ADDRESS
        BRB     FINISH                  ; FINISH UP
20$:    RSB                             ; RETURN IF NO ERROR
;
;       CHECK IEC11-B STATUS
;
STSB_ERROR:
        BLBC    R0,ERROR                ; BRANCH ON DIRECTIVE ERROR
        MOVL    IOB_STATUS,R0           ; OBTAIN I/O STATUS
        BLBC    R0,ERROR                ; BRANCH ON I/O STATUS ERROR
        RSB                             ; RETURN IF NO ERROR
;
;       DIRECTIVE ERROR CHECKING
;
TST_ERROR:
        BLBC    R0,ERROR                ; BRANCH ON DIRECTIVE ERROR
        RSB

;
;       ERROR ENTRY: DETERMINE THE PC LOCATION AND PRINT IT OUT
;
```

```
ERROR:  MOVL    (SP)+,ERRLOC            ; POP RETURN ADDRESS
        SUBL    #IEABMDEM,ERRLOC        ; SUBTRACT ROUTINE START
        PUSHR   #^M<R0>                 ; SAVE ERROR STATUS
        CALLG   ERRMSG,G^LIB$PUT_OUTPUT ; WRITE ERROR MESSAGE                 ;EJL001
        PUSHAL  LOCDSC                                                        ;EJL001
        PUSHL   ERRLOC                                                        ;EJL001
        CALLS   #2,G^FOR$CNV_OUT_Z      ; CONVERT ERROR LOCATION TO ASCII     ;EJL001
        CALLG   LOCMSG,G^LIB$PUT_OUTPUT ; WRITE ERROR LOCATION                ;EJL001
        BRB     ERRFIN                                                        ;**-5
FINISH: PUSHR   #^M<R0>                 ; SAVE STATUS
ERRFIN: $DASSGN_S CHAN = IEACHAN        ; DEASSIGN IEC11-A
        $DASSGN_S CHAN = IBACHAN        ; DEASSIGN IEC11-B
        POPR    #^M<R0>                 ; RESTORE STATUS
        RET

        .DSABL  LSB
        .PAGE
;       .SBTTL  IECAST - IEC UNSOLICITED INTERRUPT AST ROUTINES
;
;++
; FUNCTIONAL DESCRIPTION:
;
;       IF THE AST WAS CAUSED BY A SERVICE REQUEST INTERRUPT, THEN
;       THIS PROGRAM CONDUCTS A SERIAL POLL AND PRINTS THE RESULTS
;
;       IF THE AST IS NOT A SERVICE REQUEST, THEN THE PROGRAM SIMPLY
;       PRINTS THE REASON FOR THE AST
;
; CALLING SEQUENCE:
;
;       AST ROUTINE: THE ENTRY IS INAAST FOR THE IEC11-A AND INBAST FOR
;                    THE IEC11-B
;
; INPUT PARAMETERS:
;
;       4(AP) - THE AST PARAMETER FROM THE IO$_SETMODE!IO$M_UNS_AST CALL
;
;--
       .ENABL   LSB
;++
;       INAAST - UNSOLICITED IEC11-A AST ENTRY POINT
;
;--

;...    IF <Service Request>
;...    THEN
;...    <conduct a serial poll and report the results>
;...    <re-enable service request interrupts>
;...    ELSE
;...    <Report the AST parameter>
;...    ENDIF
```

```
        .ENTRY  INAAST,0
        MOVB    DEV_CHAR,AST_DEV_MSG    ; SAY THE IEC11-A DID IT
        CMPL    #IE_AST_SERVICE,4(AP)   ; SERVICE REQUEST?
        BEQL    10$                     ; YES IF EQL
        BRW     30$                     ; NO IF NOT
;
;       SERVICE REQUEST -- CONDUCT A POLL OF THE IEC11-B
;
10$:    $QIOW_S CHAN = IEACHAN,-        ; YES, CONDUCT A SERIAL POLL
                FUNC = #IO$_SER_POLL,-  ;  OF THE IEC11-B ONLY SINCE
                EFN  = #1,-             ;  THIS IS THE ONLY OTHER ACTIVE
                IOSB = IOP_STATUS,-     ;  DEVICE
                P1 = DEVBUF,-           ; SERIAL POLL DEVICE LIST
                P2 = #1,-               ; POLL ONE DEVICE
                P3 = #RESBUF            ; SAVE THE RESPONSE HERE
        BLBC    R0,20$                  ; BRANCH IF ERROR
        BLBC    IOP_STATUS,20$          ; BRANCH IF ERROR
        PUSHAL  RSPDSC                  ; CONVERT POLL BYTE TO HEX
        PUSHL   RESBUF
        CALLS   #2,G^FOR$CNV_OUT_Z                                          ;EJL001
        CALLG   POLLMSG,G^LIB$PUT_OUTPUT ; WRITE THE POLL RESULT            ;EJL001
        $QIOW_S CHAN = IEACHAN,-        ; RE-ENABLE SRQ INTERRUPTS          ;**-2
                FUNC = #IO$_SETMODE!IO$M_ENA_SRQ,-
                EFN  = #1,-             ;
                IOSB = IOP_STATUS       ;
        BLBC    R0,20$                  ; BRANCH IF ERROR
        BLBC    IOP_STATUS,20$          ; BRANCH IF ERROR
        BRB     60$                     ; RETURN
20$:    CALLG   POLL_ERR_MSG,G^LIB$PUT_OUTPUT ; SERIAL POLL ERROR           ;EJL001
        BRB     60$                     ; RETURN                            ;**-1
30$:    CMPL    #IE_AST_ERROR,4(AP)     ; HARDWARE ERROR?
        BNEQ    40$                     ; NO IF NEQ
;
;       HERE FOR HARDWARE ERROR
;
        CALLG   AST_ERR_MSG,G^LIB$PUT_OUTPUT ; YES, TELL USER               ;EJL001
        BRB     60$                     ; RETURN                            ;**-1
40$:    CMPL    #IE_AST_UNREC,4(AP)     ; UNRECOGNIZED INTERRUPT?
        BNEQ    50$                     ; NO, MUST BE STATE CHANGE
;
;       HERE FOR UNRECOGNIZED MESSAGE
;
        CALLG   AST_UNR_MSG,G^LIB$PUT_OUTPUT ; YES, TELL USER               ;EJL001
        BRB     60$                     ; RETURN                            ;**-1
;++
;       INBAST - UNSOLICITED IEC11-B AST ENTRY POINT
;                THE ONLY INTERRUPTS RECOGNIZED BY THE IEC11-B AST
;                MECHANISM ARE STATE CHANGES TO TACS OR LACS, WHICH
;                ARE REPORTED TO THE USER
;--
        .ENTRY  INBAST,0
        MOVB    DEV_CHAR+1,AST_DEV_MSG  ; SAY THE IEC11-B DID IT
;
;       HERE FOR A STATE CHANGE
;
50$:    MOVL    4(AP),R0                ; GET THE NEW STATE
        MOVAL   STATES,R1               ; GET THE STATE LIST
        MOVL    (R1)[R0],STATE_TXT      ; PUT THE STATE IN THE MESSAGE
        CALLG   AST_STATE_MSG,G^LIB$PUT_OUTPUT ; TELL THE USER WHAT HAPPENED    ;EJL001

;...    <Return>                                                            ;**-1

60$:     RET                            ; RETURN FROM AST

        .DSABL  LSB
        .END IEABMDEM                   ;END OF MODULE
```

B.2 DIFFERENCE COMPARISON OF THE PROGRAMS

This listing is a difference listing showing the change from/to entries required to convert the program above to run with the IEX-VMS-Driver.

Note: Only those changes necessary for proper program execution were made. Further changes to clarify comments and parameter names should be made to facilitate future maintenance requirements.

```
************
File USER2:[GPIB]IEABMDEM.MAR;1
   74           $IECDEF                         ; IEC CODE DEFINITIONS
   75   ;
******
File USER2:[GPIB]IEC_TO_IEX_CONV.MAR;3
   75           IEX$FUNC                        ; IEX Function Definitions
   76           RSVBYT  =       ^O123           ; Service request status byte
   77   ;
************

************
File USER2:[GPIB]IEABMDEM.MAR;1
  105   IEANAMDSC:      DESCRIPTOR      <IEA0>  ; IEC11-A NAME DESCRIPTOR
  106   IBANAMDSC:      DESCRIPTOR      <IBA0>  ; IEC11-B NAME DESCRIPTOR
  107   APRDSC:         DESCRIPTOR      <WHAT IS THE IEC11-A IEEE BUS ADDRESS (DEC)? >
  108   BPRDSC:         DESCRIPTOR      <WHAT IS THE IEC11-B IEEE BUS ADDRESS (DEC)? >
  109   STRTMSG:
******
File USER2:[GPIB]IEC_TO_IEX_CONV.MAR;3
  107   IEANAMDSC:      DESCRIPTOR      <IXA0>  ; IEX11 Unit 0 name descriptor
  108   IBANAMDSC:      DESCRIPTOR      <IXA1>  ; IEX11 Unit 1 name descriptor
  109   APRDSC:         DESCRIPTOR      <WHAT IS THE IXA0: IEEE BUS ADDRESS (DEC)? >
  110   BPRDSC:         DESCRIPTOR      <WHAT IS THE IXA1: IEEE BUS ADDRESS (DEC)? >
  111   STRTMSG:
************

************
File USER2:[GPIB]IEABMDEM.MAR;1
  206                   FUNC = #IO$_INITIALIZE!IO$M_CACS,- ;  ACTIVE) STATE
  207                   EFN  = #1,-
  208                   IOSB = IOA_STATUS
  209           BSBW    STSA_ERROR              ; CHECK FOR ERROR
  210           $QIOW_S CHAN = IEACHAN,-        ; ENABLE UNSOLICITED INTERRUPT ASTS
  211                   FUNC = #IO$_SETMODE!IO$M_UNS_AST,- ;
  212                   EFN  = #2,-
  213                   IOSB = IOA_STATUS,-     ;
  214                   ASTADR = INAAST         ;
  215                   BSBW    STSA_ERROR      ; CHECK FOR ERROR
******
File USER2:[GPIB]IEC_TO_IEX_CONV.MAR;3
  208                   FUNC = #IO$_INITIALIZE,- ;  ACTIVE) STATE
  209                   EFN  = #1,-
  210                   IOSB = IOA_STATUS,-
  211                   P1   = 55,-             ; Initialization mask
  212                   P2   = A_ADDRESS        ; IEC/IEEE bus address
  213           BSBW    STSA_ERROR              ; CHECK FOR ERROR
  214           $QIOW_S CHAN = IEACHAN,-        ; ENABLE UNSOLICITED INTERRUPT ASTS
  215                   FUNC = #IO$_SETMODE!IO$M_ATTNAST,- ;
  216                   EFN  = #2,-
  217                   IOSB = IOA_STATUS,-     ;
  218                   P1   = INAAST           ;
  219           BSBW    STSA_ERROR              ; CHECK FOR ERROR
************
************
File USER2:[GPIB]IEABMDEM.MAR;1
  221                   FUNC = #IO$_SETMODE!IO$M_ENA_SRQ,-
  222                   EFN  = #1,-
  223                   IOSB = IOA_STATUS
```

```
  224           BSBW    STSA_ERROR              ; CHECK FOR ERROR
******
File USER2:[GPIB]IEC_TO_IEX_CONV.MAR;3
  225                   FUNC = #IO$_SETEVENT,-
  226                   EFN  = #1,-
  227                   IOSB = IOA_STATUS,-
  228                   P1   = 1                ; Enable service request events
  229           BSBW    STSA_ERROR              ; CHECK FOR ERROR
************
************
File USER2:[GPIB]IEABMDEM.MAR;1
  231                   IOSB = IOB_STATUS
  232           BSBW    STSB_ERROR              ; CHECK FOR ERROR
  233           $QIOW_S CHAN = IBACHAN,-        ; ENABLE UNSOLICITED INTERRUPT ASTS
  234                   FUNC = #IO$_SETMODE!IO$M_UNS_AST,-
  235                   EFN  = #3,-
  236                   IOSB = IOB_STATUS,-     ;
  237                   ASTADR = INBAST         ;
  238           BSBW    STSB_ERROR              ; CHECK FOR ERROR
******
File USER2:[GPIB]IEC_TO_IEX_CONV.MAR;3
  236                   IOSB = IOB_STATUS,-
  237                   P1   = 35,-             ; Initialization mask
  238                   P2   = B_ADDRESS        ; IEC/IEEE bus address
  239           BSBW    STSB_ERROR              ; CHECK FOR ERROR
  240           $QIOW_S CHAN = IBACHAN,-        ; ENABLE UNSOLICITED INTERRUPT ASTS
  241           FUNC = #IO$_SETMODE!IO$M_ATTNAST,-
  242                   EFN  = #3,-
  243                   IOSB = IOB_STATUS,-     ;
  244                   P1   = INBAST           ;
  245           BSBW    STSB_ERROR              ; CHECK FOR ERROR
************

************
File USER2:[GPIB]IEABMDEM.MAR;1
  242   ;...    <Transfer a string from the IEC11-A to the IEC11-B>
******
File USER2:[GPIB]IEC_TO_IEX_CONV.MAR;3
  249   ;...    <Enable addressed events>
  250   ;
  251           $QIOW_S CHAN = IBACHAN,-        ; Enable addressed events
  252                   FUNC = #IO$_SETEVENT,-
  253                   EFN  = #1,-
  254                   IOSB = IOB_STATUS,-
  255                   P1   = 6                ; Addressed events
  256           BSBW    STSB_ERROR              ; Check for error
  257   ;
  258   ;...    <Transfer a string from the IEC11-A to the IEC11-B>
************
```

```
************
File USER2:[GPIB]IEABMDEM.MAR;1
  247                   FUNC = #IO$_READVBLK!IO$M_REQSRV,- ; AS LACS AND REQUEST
  248                   IOSB = IOB_STATUS,-     ;  SERVICE AFTERWARDS
******
File USER2:[GPIB]IEC_TO_IEX_CONV.MAR;3
  263                   FUNC = #IO$_READVBLK,-  ;  AS LACS AND REQUEST
  264                   IOSB = IOB_STATUS,-     ;  SERVICE AFTERWARDS
************
************
File USER2:[GPIB]IEABMDEM.MAR;1
  256                   FUNC = #IO$_WRITEVBLK!IO$M_COMMAND,- ; IEC11-A AS TALKER AND
  257                   EFN  = #1,-             ; THE IEC11-B AS LISTENER
  258                   IOSB = IOA_STATUS,-     ;
******
File USER2:[GPIB]IEC_TO_IEX_CONV.MAR;3
  272                   FUNC = #IO$_COMMANDS,-  ;  IEC11-A AS TALKER AND
  273                   EFN  = #1,-             ;  THE IEC11-B AS LISTENER
  274                   IOSB = IOA_STATUS,-     ;
************
************
File USER2:[GPIB]IEABMDEM.MAR;1
  271           CALLG   RESMSG,G^LIB$PUT_OUTPUT ; WRITE THE RESULTS                     ;EJL001
******
File USER2:[GPIB]IEC_TO_IEX_CONV.MAR;3
  287           $QIOW_S CHAN = IBACHAN,-        ; Generate service request
  288                   FUNC = #IO$_SERVICE,-   ;
  289                   EFN  = #4,-             ;
  290                   IOSB = IOB_STATUS,-     ;
  291                   P1 = RSVBYT             ; Service request status byte
  292           BSBW    STSB_ERROR              ; Check for error
  293           CALLG   RESMSG,G^LIB$PUT_OUTPUT ; WRITE THE RESULTS                     ;EJL001
************
************
File USER2:[GPIB]IEABMDEM.MAR;1
  280                   FUNC = #IO$_WRITEVBLK!IO$M_COMMAND,- ; IEC11-A AS LISTENER
  281                   EFN  = #1,-             ; AND THE IEC11-B AS TALKER
******
File USER2:[GPIB]IEC_TO_IEX_CONV.MAR;3
  302                   FUNC = #IO$_COMMANDS,-  ; IEC11-A AS LISTENER
  303                   EFN  = #1,-             ; AND THE IEC11-B AS TALKER
************
************
File USER2:[GPIB]IEABMDEM.MAR;1
  294                   FUNC = #IO$_WRITEVBLK!IO$M_REQSRV,- ; REQUEST SERVICE WHEN
  295                   IOSB = IOB_STATUS,-     ;    DONE
******
File USER2:[GPIB]IEC_TO_IEX_CONV.MAR;3
  316                   FUNC = #IO$_WRITEVBLK,- ;    REQUEST SERVICE WHEN
  317                   IOSB = IOB_STATUS,-     ;    DONE
************
************
File USER2:[GPIB]IEABMDEM.MAR;1
  302           CALLG   RESMSG,G^LIB$PUT_OUTPUT ; WRITE THE RESULTS                     ;EJL001
******
File USER2:[GPIB]IEC_TO_IEX_CONV.MAR;3
  324           $QIOW_S CHAN = IBACHAN,-        ; Generate service request
  325                   FUNC = #IO$_SERVICE,-   ;
  326                   EFN  = #4,-             ;
  327                   IOSB = IOB_STATUS,-     ;
  328                   P1 = RSVBYT             ; Service request status byte
  329           BSBW    STSB_ERROR              ; Check for error
  330           CALLG   RESMSG,G^LIB$PUT_OUTPUT ; WRITE THE RESULTS                     ;EJL001
************
************
File USER2:[GPIB]IEABMDEM.MAR;1
  393           MOVB    DEV_CHAR,AST_DEV_MSG    ; SAY THE IEC11-A DID IT
  394           CMPL    #IE_AST_SERVICE,4(AP)   ; SERVICE REQUEST?
  395           BEQL    10$                     ; YES IF EQL
  396           BRW     30$                     ; NO IF NOT
```

```
  397   ;
******
File USER2:[GPIB]IEC_TO_IEX_CONV.MAR;3
  421           $QIOW_S CHAN = IEACHAN,-        ; Re-enable attention ASTs
  422                   FUNC = #IO$_SETMODE!IO$M_ATTNAST,- ;
  423                   EFN  = #2,-
  424                   IOSB = IOA_STATUS,-     ;
  425                   P1   = INAAST           ;
  426           BSBW    STSA_ERROR              ; CHECK FOR ERROR
  427           MOVB    DEV_CHAR,AST_DEV_MSG    ; SAY THE IEC11-A DID IT
  428           CMPL    #1,4(AP)                ; SERVICE REQUEST?
  429           BEQL    10$                     ; YES IF EQL
  430           BRW     60$                     ; NO IF NOT
  431   ;
************
************
File USER2:[GPIB]IEABMDEM.MAR;1
  413           $QIOW_S CHAN = IEACHAN,-        ; RE-ENABLE SRQ INTERRUPTS              ;**-2
  414                   FUNC = #IO$_SETMODE!IO$M_ENA_SRQ,-
  415                   EFN  = #1,-             ;
  416                   IOSB = IOP_STATUS       ;
  417           BLBC    R0,20$                  ; BRANCH IF ERROR
  418           BLBC    IOP_STATUS,20$          ; BRANCH IF ERROR
  419           BRB     60$                     ; RETURN
  420   20$:    CALLG   POLL_ERR_MSG,G^LIB$PUT_OUTPUT ; SERIAL POLL ERROR             ;EJL001
  421           BRB     60$                     ; RETURN                                ;**-1
  422   30$:    CMPL    #IE_AST_ERROR,4(AP)     ; HARDWARE ERROR?
  423           BNEQ    40$                     ; NO IF NEQ
  424   ;
  425   ;       HERE FOR HARDWARE ERROR
  426   ;
  427           CALLG   AST_ERR_MSG,G^LIB$PUT_OUTPUT ; YES, TELL USER                 ;EJL001
  428           BRB     60$                     ; RETURN                                ;**-1
  429   40$:    CMPL    #IE_AST_UNREC,4(AP)     ; UNRECOGNIZED INTERRUPT?
  430           BNEQ    50$                     ; NO, MUST BE STATE CHANGE
  431   ;
  432   ;       HERE FOR UNRECOGNIZED MESSAGE
  433   ;
  434           CALLG   AST_UNR_MSG,G^LIB$PUT_OUTPUT ; YES, TELL USER                 ;EJL001
  435           BRB     60$                     ; RETURN                                ;**-1
******
File USER2:[GPIB]IEC_TO_IEX_CONV.MAR;3
  447           BRW     60$                     ; Return
  448   20$:    CALLG   POLL_ERR_MSG,G^LIB$PUT_OUTPUT ; SERIAL POLL ERROR             ;EJL001
  449           BRB     60$                     ; RETURN                                ;**-1
************
************
File USER2:[GPIB]IEABMDEM.MAR;1
  447   50$:    MOVL    4(AP),R0                ; GET THE NEW STATE
  448           MOVAL   STATES,R1               ; GET THE STATE LIST
******
File USER2:[GPIB]IEC_TO_IEX_CONV.MAR;3
  461   50$:    MOVZBL  4(AP),R0                ; GET THE NEW STATE
  462           MOVAL   STATES,R1               ; GET THE STATE LIST
************
```

```
************
File USER2:[GPIB]IEABMDEM.MAR;1
  451   ;...    <Return>                                                       ;**-1
  452   60$:    RET                            ; RETURN FROM AST
******
File USER2:[GPIB]IEC_TO_IEX_CONV.MAR;3
  465           $QIOW_S CHAN = IBACHAN,-        ; Re-enable attention ASTs
  466                   FUNC = #IO$_SETMODE!IO$M_ATTNAST,- ;
  467                   EFN  = #3,-
  468                   IOSB = IOB_STATUS,-     ;
  469                   P1   = INBAST           ;
  470           BSBW    STSB_ERROR              ; Check for error
  471           $QIO_S  CHAN = IBACHAN,-        ; Issue Auxiliary DACR command
  472                   FUNC = #IO$_AUXILIARY,- ; NOTE - This cannot be a $QIOW_S
  473                   P1   = 1                ; Release DAC holdoff
  474           BSBW    TST_ERROR               ; Check for error
  475   ;
  476   ;...    <Return>                                                       ;**-1
  477   ;
  478   60$:    RET                            ; RETURN FROM AST
************

Number of difference sections found: 16
Number of difference records found: 103
```

C IEC 625/IEEE-488 BUS COMMANDS

The following table represents the valid IEC 625/IEEE-488 bus commands.

Hex	Octal	Decimal	ASCII	Message
00	000	0	NUL	
01	001	1	SOH	GTL
02	002	2	STX	
03	003	3	ETX	
04	004	4	EOT	SDC
05	005	5	ENQ	PPC
06	006	6	ACK	
07	007	7	BEL	
08	010	8	BS	GET
09	011	9	HT	TCT
0A	012	10	LF	
0B	013	11	VT	
0C	014	12	FF	
0D	015	13	CR	
0E	016	14	SO	
0F	017	15	SI	
10	020	16	DLE	
11	021	17	DC1	LLO
12	022	18	DC2	
13	023	19	DC3	
14	024	20	DC4	DCL
15	025	21	NAK	PPU
16	026	22	SYN	
17	027	23	ETB	
18	030	24	CAN	SPE
19	031	25	EM	SPD
1A	032	26	SUB	
1B	033	27	ESC	
1C	034	28	FS	
1D	035	29	GS	
1E	036	30	RS	
1F	037	31	US	

Hex	Octal	Decimal	ASCII	Message
20	040	32	SP	MLA 0
21	041	33	!	MLA 1
22	042	34	"	MLA 2
23	043	35	#	MLA 3
24	044	36	$	MLA 4
25	045	37	%	MLA 5
26	046	38	&	MLA 6
27	047	39	’	MLA 7
28	050	40	(	MLA 8
29	051	41	)	MLA 9
2A	052	42	*	MLA 10
2B	053	43	+	MLA 11
2C	054	44	,	MLA 12
2D	055	45	-	MLA 13
2E	056	46	.	MLA 14
2F	057	47	/	MLA 15
30	060	48	0	MLA 16
31	061	49	1	MLA 17
32	062	50	2	MLA 18
33	063	51	3	MLA 19
34	064	52	4	MLA 20
35	065	53	5	MLA 21
36	066	54	6	MLA 22
37	067	55	7	MLA 23
38	070	56	8	MLA 24
39	071	57	9	MLA 25
3A	072	58	:	MLA 26
3B	073	59	;	MLA 27
3C	074	60	<	MLA 28
3D	075	61	=	MLA 29
3E	076	62	>	MLA 30
3F	077	63	?	UNL
40	100	64	@	MTA 0
41	101	65	A	MTA 1
42	102	66	B	MTA 2
43	103	67	C	MTA 3
44	104	68	D	MTA 4
45	105	69	E	MTA 5
46	106	70	F	MTA 6

Hex	Octal	Decimal	ASCII	Message
47	107	71	G	MTA 7
48	110	72	H	MTA 8
49	111	73	I	MTA 9
4A	112	74	J	MTA 10
4B	113	75	K	MTA 11
4C	114	76	L	MTA 12
4D	115	77	M	MTA 13
4E	116	78	N	MTA 14
4F	117	79	O	MTA 15
50	120	80	P	MTA 16
51	121	81	Q	MTA 17
52	122	82	R	MTA 18
53	123	83	S	MTA 19
54	124	84	T	MTA 20
55	125	85	U	MTA 21
56	126	86	V	MTA 22
57	127	87	W	MTA 23
58	130	88	X	MTA 24
59	131	89	Y	MTA 25
5A	132	90	Z	MTA 26
5B	133	91	[	MTA 27
5C	134	92	\	MTA 28
5D	135	93	]	MTA 29
5E	136	94	^	MTA 30
5F	137	95	_	UNT
60	140	96	‘	MSA,PPE
61	141	97	a	MSA,PPE
62	142	98	b	MSA,PPE
63	143	99	c	MSA,PPE
64	144	100	d	MSA,PPE
65	145	101	e	MSA,PPE
66	146	102	f	MSA,PPE
67	147	103	g	MSA,PPE
68	150	104	h	MSA,PPE
69	151	105	i	MSA,PPE
6A	152	106	j	MSA,PPE
6B	153	107	k	MSA,PPE
6C	154	108	l	MSA,PPE
6D	155	109	m	MSA,PPE

Hex	Octal	Decimal	ASCII	Message
6E	156	110	n	MSA,PPE
6F	157	111	o	MSA,PPE
70	160	112	p	MSA,PPD
71	161	113	q	MSA,PPD
72	162	114	r	MSA,PPD
73	163	115	s	MSA,PPD
74	164	116	t	MSA,PPD
75	165	117	u	MSA,PPD
76	166	118	v	MSA,PPD
77	167	119	w	MSA,PPD
78	170	120	x	MSA,PPD
79	171	121	y	MSA,PPD
7A	172	122	z	MSA,PPD
7B	173	123	{	MSA,PPD
7C	174	124	|	MSA,PPD
7D	175	125	}	MSA,PPD
7E	176	126	~	MSA,PPD
7F	177	127	DEL	

D IEX-VMS-Driver V4.2 KIT CONTENTS

The following list represents the contents and file descriptions of the IEX-VMS-Driver kit.

File	Description
KITINSTAL.COM	IEX Installation Command File
IEX$042.RELEASE_NOTES	IEX-VMS-Driver V4.2 Release Notes
IXDRIVER.COM	Supplement command file to link driver
IXDRIVER.OBJ	Object code for driver (VMS V5.x)
IXDRIVER_V4X.OBJ	Object code for driver (VMS V4.x)
IXDRIVER.OPT	Linker Option file for driver
IEX.EXE	IEX exerciser
IEX.OPT	Linker Option file for IEX exerciser
IEX.COM	Command file to build IEX exerciser
IEX$DEMO.COM	Indirect command file to demonstrate the features of the IEX11
IEX.FOR	IEX exerciser source module
IEX$DEX.FOR	IEX exerciser control program source module
UETIX1100.EXE	IEX UETP program (VMS V5.x)
UETIX1100_V4X.EXE	IEX UETP program (VMS V4.x)
IEX$IVP.EXE	IEX installation verification procedure
IEX$IVP.MAR	IEX installation verification procedure source module
IEX$IVP.COM	Command file to build IEX$IVP
IEX$IVP.OPT	Linker Option file for IEX$IVP
IEX$SUI.OBJ	Simplified User Interface object code
IEX$SUI.MAR	Simplified User Interface source module
IEX$SUI.COM	Command file to build IEX$SUI
IEX$FUNC.FOR	Function definition include file for FORTRAN
IEX$FUNC.MAR	Function definition include file for MACRO-32
IEX$FUNC.MLB	Function definition macro library (from IEX$FUNC.MAR)
IEX$FUNC.BAS	Function definition include file for BASIC
IEX$FUNC.PAS	Function definition include file for PASCAL
IEX$DEMO.OPT	Linker Option file for IEX demonstration programs
IEX$DEMOB.FOR	Subprocess demonstration program FORTRAN source module
IEX$DEMOB.EXE	Subprocess demonstration program FORTRAN executable
IEX$DEMOAB.BAS	Demonstration program BASIC source module
IEX$DEMOAB.EXE	Demonstration program BASIC executable
IEX$DEMOAF.FOR	Demonstration program FORTRAN source module

IEX-VMS-Driver V4.2 KIT CONTENTS

File	Description
IEX$DEMOAF.EXE	Demonstration program FORTRAN executable
IEX$DEMOAM.MAR	Demonstration program MACRO-32 source module
IEX$DEMOAM.EXE	Demonstration program MACRO-32 executable (VMS V5.x)
IEX$DEMOAM_V4X.EXE	Demonstration program MACRO-32 executable (VMS V4.x)
IEX$DEMOAP.PAS	Demonstration program PASCAL source module
IEX$DEMOAP.EXE	Demonstration program PASCAL executable
IEX$DEMOAS.FOR	Demonstration SUI program FORTRAN source module
IEX$DEMOAS.EXE	Demonstration SUI program FORTRAN executable
IEX$DEMONS.COM	Command file to assemble and build demonstration programs

Index

A

C

D

E

H

I

I

L

N

P

R

S

T

U