Instrument Bus Subroutines Programmer's Reference Manual

AA–5613C–TC and AD–5613C–T1

August 1982

This document describes how to use the FORTRAN Instrument Bus Subroutines to support either the IBV11–A or IB11 Instrument Bus Interface between a PDP–11 system and the IEEE 488–1978 General Purpose Instrument Bus.

This document updates the Instrument Bus Subroutines Programmer's Reference Manual, order number AA–5613C–TC.

OPERATING SYSTEM:	RT–11 V4.0
SOFTWARE:	IBS V2.1 FORTRAN IV V2.5

Software and manuals should be ordered by title and order number. In the United States, send orders to the nearest distribution center. Outside the United States, orders should be directed to the nearest DIGITAL Field Sales Office or representative.

Northeast/Mid–Atlantic Region	**Central Region**	**Western Region**
Digital Equipment Corporation PO Box CS2008 Nashua, New Hampshire 03061 Telephone:(603)884–6660	Digital Equipment Corporation Accessories and Supplies Center 1050 East Remington Road Schaumburg, Illinois 60195 Telephone:(312)640–5612	Digital Equipment Corporation Accessories and Supplies Center 632 Caribbean Drive Sunnyvale, California 94086 Telephone:(408)734–4915

digital equipment corporation • marlboro, massachusetts

First Printing, November 1978
Revised, June 1980
Revised, October 1981
Updated, August 1982

Contents

Preface

Manual Objectives

This manual describes FORTRAN Instrument Bus (IB) Subroutines that you can use to transfer data between the PDP–11 computer and instruments on the IEEE Standard 488–1978 General Purpose Instrument Bus. The IBV11–A or IB11 Instrument Bus Interface is used to connect the computer to the IEEE bus. You can use these subroutines on any PDP–11 system under Version 4.0 of the RT–11 Single Job (SJ), Foreground/Background (FB), or Extended Memory (XM) operating systems.

The capabilities of the IEEE Bus are specified in the IEEE publication entitled IEEE Standard Digital Interface for Programmable Instrumentation (488–1978).

To use this manual you should be familiar with a laboratory, should understand the capabilities and operation of all instruments in your system, and should have access to instrument manufacturers' documentation. You should also be familiar with the RT–11 SJ, FB, or XM operating system, and with the FORTRAN IV programming language.

You can use this manual whether you are just beginning to use the IB subroutines in programming, or whether you have more advanced programming skills.

Manual Structure

Chapter 1 describes the IEEE Bus. It also explains how the IB subroutines control the activities of the bus through the IBV11–A/IB11 Instrument Bus Interface.

Chapter 2 explains how the IEEE bus works.

Chapter 3 explains how to install the IBV11–A instrument bus interface on an LSI–11 system and tells where to find information to install the interface on other PDP–11 systems.

Chapter 4 explains how to install, test, and use the IB subroutines.

Chapter 5 explains the notational conventions, addressing techniques and formats needed to understand and use the IB subroutines. The chapter also describes the IB subroutines that are used in data transmission, and which are the subroutines required most often in a user's program.

Chapters 6 through 9 explain the IB subroutines that control activities of the instrument bus other than data transmission. These activities include polling routines that check for instrument service requests and routines that handle errors in the IB subroutines.

Chapter 10 contains IB subroutines used in more advanced programming.

The Appendixes provide information about ASCII character codes, command mnemonics, installing additional IBV11–A units, and IB subroutine formats. A sample instrument address form is provided.

Related Documents

The following documents provide more information about the RT–11 operating system and the FORTRAN–IV programming language.

Software Manual	**Order Number**
Introduction to RT–11	AA–5281B–TC
RT–11 Programmer's Reference Manual	AA–H378A–TC
RT–11 Software Support Manual	AA–H379A–TC
RT–11 System User's Guide	AA–5279B–TC
RT–11/RSTS/E FORTRAN IV User's Guide	AA–5749B–TC
RT–11 Installation and System Generation Guide	AA–H376A–TC

Hardware Manual	**Order Number**
IB11 UNIBUS to IEEE Instrument Bus Interface Installation Manual	EK–IB11A–IN–001
IBV11–A LSI–11/Instrument Bus Interface User's Manual	EK–IBV11–UG–001
Microcomputer Processor Handbook	EB–15836–18/79

Documentation Conventions

The following conventions apply to this manual:

1. In programming examples, all information the computer prints appears in black. All commands and responses you type appear in red.
2. RET means you must press the RETURN key on your terminal.
3. CTRL/x represents the CTRL key and another key (represented here by x) which control some of the computer's functions. To perform one of these functions, hold down the CTRL key and type the other letter.
4. In examples of commands or file names, capital letters represent actual commands, file names or file types which you must type as shown. Lowercase letters mean that you must supply a name.
5. ASCII characters are the usual data format used to transmit data and programmed instructions between instruments as well as between instruments and the IBV11–A/IB11 instrument bus interface. Therefore, this manual refers to bytes of data as "characters."

Chapter 1
Fundamental IEEE Bus Concepts

1.1 What is the IEEE Bus?

The IEEE bus is a cable consisting of 16 wires, which are called *bus lines*. These bus lines are shared by MINC and all instruments on the bus. The cable, connectors to the cable, and electrical requirements are defined by standards set by the Institute of Electrical and Electronic Engineers.

The IEEE bus allows a computer to communicate with and control a variety of instruments. By using strings of characters called *messages*, the computer can send an instrument a message that tells the instrument what to do. The instrument can send back a message about data it has gathered.

The standard describes each line of the bus and specifies exactly when and how an instrument may use that line. Eight of the lines, called the *data lines*, are used to encode the messages sent on the bus. Three more lines, called the *handshake lines*, are used to make certain that each character sent is received. The remaining five lines are for general bus management. The purpose of each of these 16 lines is discussed in more detail later.

1.2 What are the FORTRAN IB Routines?

The FORTRAN IB routines described in this manual control the activities of the IEEE Instrument Bus. Through the interface, these routines communicate with instruments connected to the IEEE bus. They specify instruments as talkers or listeners and control the transmission of data on the bus.

Instruments can be polled to determine their status, using one of two polling techniques: serial polling or parallel polling. A serial poll checks the status of one instrument at a time. A parallel poll checks the status of up to eight instruments simultaneously.

Instruments on the bus can request service from the IB routines by setting one of the bus's 16 lines, which is reserved for this purpose. You can designate a FORTRAN subroutine that will be called automatically whenever an instrument requests service.

When an error occurs in an IB routine, an error message is typed, indicating the type of error. An error will normally stop your program, but you can specify that a given type of error will not stop your program.

1.3 Talker, Listener, Controller

Instruments play well-defined roles on the bus. An instrument sending a message is called a *talker.* Only one instrument may *talk* at any one time. An instrument receiving a message is called a *listener.* Any number of instruments can *listen* to the message being sent by the talker. Instruments can be talkers only, listeners only, or talkers and listeners. The user's guide for your instrument will tell you what your instruments are.

The computer is called the *controller* of the bus. As such, it tells bus instruments when to talk and when to listen. No instrument can ever talk or listen unless told to do so by the computer. The computer controls all bus activity, and it must be the only controller of the bus. This means that no other device not even another computer, can be a controller on this IEEE bus. The computer can make itself a talker or listener, however, it listens to all traffic on the bus.

For example, suppose your IEEE bus system consists of a computer, a multimeter, and a signal generator. You want the computer to receive from the multimeter a message that reports a voltage reading and then send the signal generator a message that causes it to generate a signal based on the voltage reading. To receive the voltage reading, the computer tells the multimeter to be the talker, and the computer itself is the listener. The computer tells the signal generator to neither talk nor listen, so the signal generator ignores the message that the multimeter sends to the computer (see Figure 1–1). To send instructions for the signal output, the computer tells the signal generator to be a listener, and the computer itself is the talker. The computer tells the multimeter to neither talk nor listen, so the multimeter ignores the message that the computer sends to the signal generator (see Figure 1–2).

Figure 1–1: Computer Receives a Message

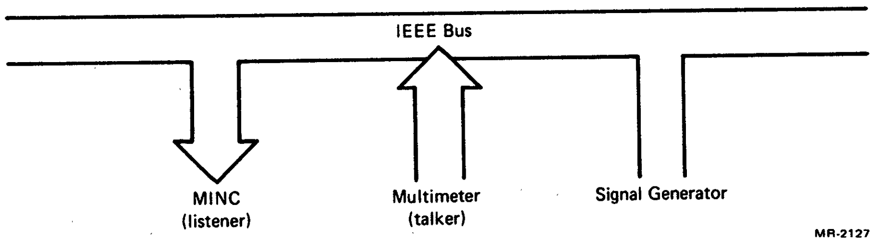
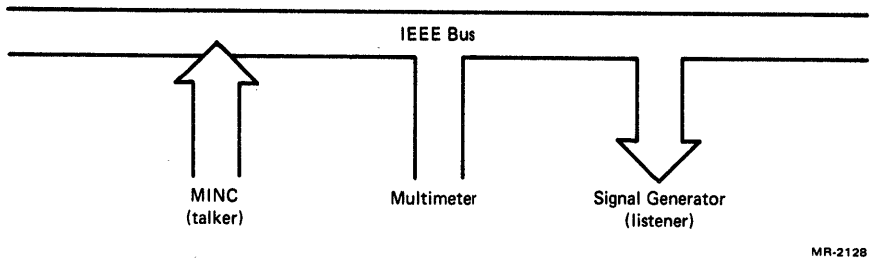


Figure 1–2: Computer Sends a Message



1.3.1 Instrument Addresses

Each instrument on the bus has a number between 0 and 30 that the computer uses to identify the instrument when the computer tells the instrument to either talk or listen. This number is the instrument's *address* and can be set with switches located on the instrument itself. Chapter 3 describes these switches and how to set them. Before you can use the IEEE routines, you must know the addresses of the instruments on your IEEE bus. In the previous example, you might set the multimeter to have an address of 1, and the signal generator to have an address of 2. An instrument's address is also called its *primary address.*

Some bus instruments have different functions or parts that the computer can specify by using a *secondary address* in addition to the instrument's primary address. Secondary addresses are in the range 0 to 30, though in order to distinguish them from primary addresses, they are specified in IEEE bus routines by numbers in the range 200 to 230. Each instrument's designer defines the meanings of any secondary addresses the instrument recognizes. For example, when you tell the multimeter above to talk, it might report a voltage reading if you specify secondary address 2, and a resistance reading if you specify secondary address 1.

We suggest that you write down the address of each bus instrument, along with any secondary addresses and what they specify, on a form such as the one shown in Figure 1–3. Be sure that no two instruments have the same primary IEEE bus address.

Figure 1–3: Sample Address Record

Instrument	Address	Secondary Addresses
Multimeter	1	1 resistance
		2 amperage
		3 voltage
Signal Generator	2	none

MR-2126

1.3.2 The Interface

Figure 1–4: An Instrument's Interface

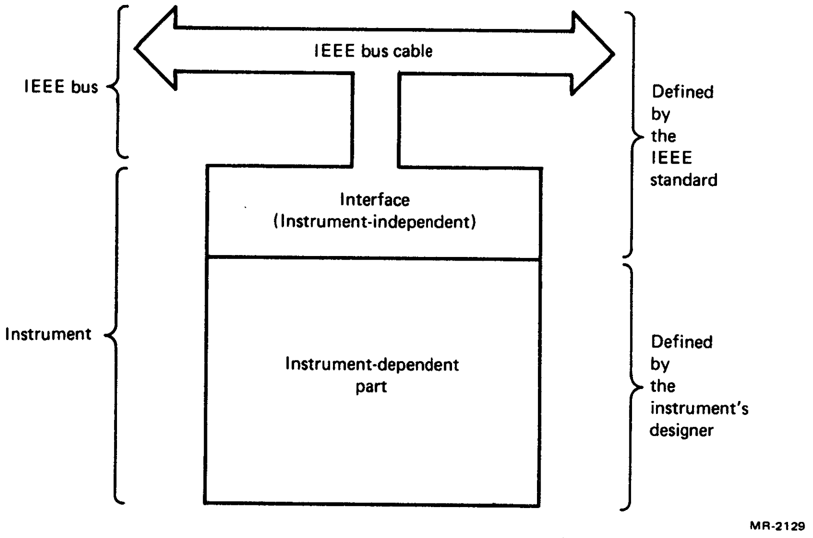
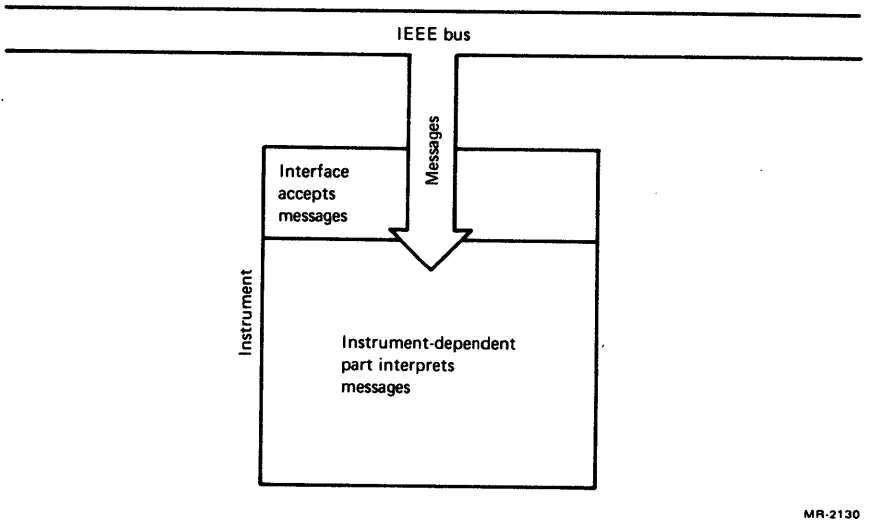

IEEE bus
IEEE bus cable
Defined by the IEEE standard
Interface (Instrument-independent)
Instrument
Instrument-dependent part
Defined by the instrument's designer
MR-2129

Figure 1–5: An Instrument Listens

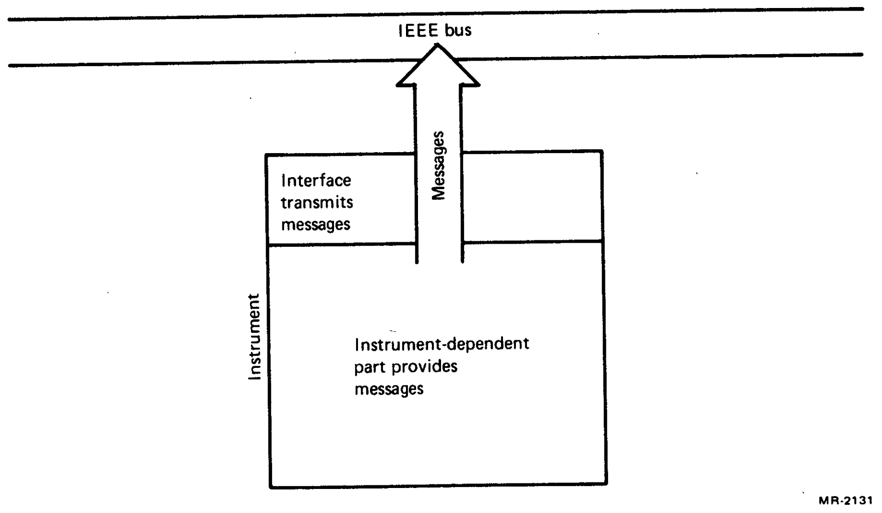

IEEE bus
Messages
Interface accepts messages
Instrument
Instrument-dependent part interprets messages
MR-2130

Part of each instrument on the IEEE bus is defined by the IEEE standard and is thus not instrument-dependent. This part is called the instrument's *interface* to the bus. The rest of the instrument is not defined by the standard but by the instrument's designer. These parts are illustrated in Figure 1–4. Because every instrument on the IEEE bus has an interface, the bus is sometimes called the interface bus.

Only an instrument's interface interacts directly with the bus. Messages are interpreted by the instrument-dependent part of the instrument, but they are sent and received through the interface. When the computer tells the instrument to listen, the instrument's interface passes any subsequent messages sent on the bus to the instrument-dependent part of the instrument. This is illustrated in Figure 1–5.

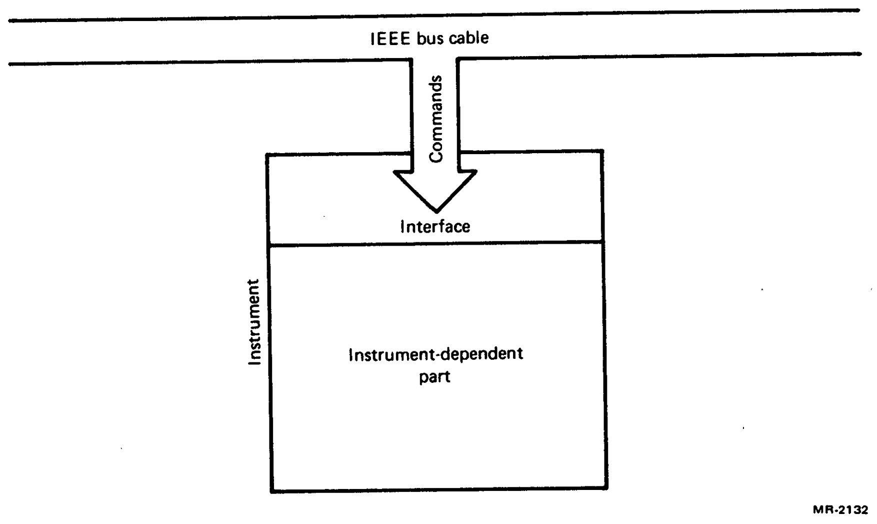
When the computer tells the instrument to talk, the instrument's interface transmits messages from the instrument-dependent part of the instrument. This is illustrated in Figure 1–6.

Figure 1–6: An Instrument Talks


IEEE bus
Messages
Interface transmits messages
Instrument
Instrument-dependent part provides messages
MR-2131

The computer controls the bus by sending *commands*, which are instructions to the interfaces on the bus. This is illustrated in Figure 1–7. Like messages, commands are sent as characters on the data lines but, unlike messages, commands are intercepted and interpreted by the interface, not passed to the instrument-dependent part of the instrument. The interface interprets each command according to the meaning defined for that command by the IEEE standard. Only the computer can send commands. The computer uses commands to tell instruments to talk or listen.

Figure 1–7: An Instrument Accepts Commands from the Computer



1.4 Message Routines

The routines described in this section control message transmission. In the IBSEND routine, the computer transmits a message to an instrument on the bus (the computer sends); in the IBRECV routine, an instrument on the bus transmits a message to the computer (the computer receives). Since message transmission between IEEE bus instruments and the computer is the main purpose of the bus, IBSEND and IBRECV are the most used and most important IEEE bus routines.

1.4.1 Messages

The contents of each message string and the effect it has or the information it reports are as varied as the types of instruments on the bus. The IEEE standard only defines *how* instruments communicate, not *what* they communicate. For this reason, the user's guide for each instrument is an essential source of information when you use the IEEE routines. It helps you decide what strings to send to that instrument and tells you what strings you should expect to receive from it.

For example, the multimeter above might send back a reading with these characters:

V+4.382E+01

where the "V" indicates that this was a voltage reading (not an amperage or resistance reading) and the other characters indicate a measurement of 43.82 volts. Based on this reading, your program might tell the signal generator to generate a 43.8 volt signal at 1250 Hz. The message telling the signal generator to do this might be:

V43.8F1250

where "V43.8" means "43.8 volts" and "F1250" means "at a frequency of 1250 Hz."

1.4.2 Sending Messages

The IBSEND routine sends a message to one or more instruments on the IEEE bus. The computer itself is the talker and sends a message string specified by your program to the listeners specified by your program. Not all instruments are able to listen.

Each instrument that can listen has a *vocabulary* of characters and a syntax that are meaningful to it. Because vocabularies differ from instrument to instrument, you usually send each message to only one instrument. The user's guide for a particular instrument lists the characters the instrument recognizes and the effect of each character. The message you send to an instrument depends on the effect you want and which character or characters cause that effect.

One of the five general management bus lines is known as the End or Identify (EOI) line. One of its functions is to allow the talker to set it while sending the last character of its current message and thus indicate the end of that message to the listeners. Some instruments do not act on any message characters until the End or Identify (EOI) line is set. The IBSEOI routine sets this bus line when it sends the last character of the string specified by your program. The EOI line could also be set by common terminating characters.

1.4.3 Receiving Messages

The IBRECV routine receives a message from an instrument on the IEEE bus. The computer is a listener and stores the message string sent by the talker you specify. Not all instruments are able to talk. For example, the signal generator might be able only to listen.

The meaning and format of the string is determined by the instrument's designer. The user's guide for an instrument tells you what information the instrument sends and the format of its messages.

For example, a statement to take a voltage reading from the multimeter above is:

```
CALL IBRECV (READ1,15,METER)
```

where READ1 is the name of the array where the information is stored, 15 is the maximum number of characters to be stored in the array, and METER is the listener address specification.

The computer knows the message is complete when one of the following three conditions occurs:

1. The talker sets the EOI bus line while sending a character. For example, the multimeter above might set the EOI line while it sends the character "1."
2. The talker sends a *terminator*, a character the computer recognizes as an end-of-message indicator. There are two types of terminators, those recognized by the computer and those recognized by the instruments. Normally, the computer recognizes a carriage return or a line feed as a terminator, but you can change which characters are terminators by using the IBTERM routine. For example, the multimeter might follow the "1" character with a carriage return character. Note that if it did this, it could not set the EOI line while sending the "1." Depending on how it was designed, an instrument might or might not set the EOI line while sending the carriage return character. Your instrument manual will tell you what character your instrument recognizes as a terminator.

3. The number of characters sent in this message reaches a limit set by your program in the IBRECV statement. For example, if the multimeter is not designed to set the EOI line or send a terminator, your program should specify the number of characters in the string as the maximum number of characters in the message.

NOTE

If the instrument sends more characters than the limit set by the program, the remaining characters are not lost by the system. Any further transmissions by the talker are delayed until the program takes some action to retrieve the remaining characters.

1.4.4 Transferring Messages

The IBXFER routine supervises the transfer of a message between instruments. Even though it does not store the message, the computer does listen so that it can know when the message is complete. As in the IBRECV routine, the computer knows the message is complete when the talker sets the EOI line, sends a terminating character, or sends a message whose length reaches a limit set by your program in the IBXFER routine.

The IBXFER routine should only be used with compatible instruments; the vocabulary of the listener you specify should be compatible with that of the talker you specify.

1.5 Triggering

The IBGET routine triggers one or more instruments to start their basic operations. Each instrument's designer defines what that instrument does when triggered. Many instruments take a reading when triggered; the value of this reading is sent when your program asks for it with the IBRECV routine.

IBGET is often used to trigger a single instrument, but it can also trigger many instruments simultaneously. For example, you might first want to use the IBSEND routine to set the ranges on a voltmeter, a temperature probe, and an ohmmeter. Later, your program can trigger all three of these instruments to take readings simultaneouly and can then obtain the respective readings one at a time by using IBRECV statements.

1.6 Status

An instrument on the IEEE bus can report information about its current status to the computer. Though the type of status information reported depends on the particular instrument, the procedures to report status are part of the IEEE standard and are the same for all instruments.

As controller of the IEEE bus, the computer can ask instruments for their status by conducting either a *serial poll* or a *parallel poll.* In a serial poll, instruments report status information, one at a time, on the bus's data lines. Since the bus has eight data lines, each instrument can report up to eight bits of status information. In a parallel poll, up to eight instruments can simultaneously report status information on the bus's data lines. Each instrument can use only one data line and can thus report only one bit of status information. The bit of information reported in a parallel poll is not necessarily related to any of the information reported in a serial poll. A particular instrument could respond to one, both, or neither of these types of polls. The user's guide for a particular instrument tells you which polls that instrument can respond to and what status information it reports.

There is one type of status information an instrument can tell the computer without having to wait to be polled. It can tell the computer that it needs service. It issues this *service request* by setting a bus line reserved for this purpose, the SRQ (service request) bus line. If your program ignores the service request, the instrument can take no further initiative. Setting the SRQ line is the only action on the IEEE bus that instruments can initiate independently of the computer.

Since bus lines are shared by all instruments, the computer doesn't know which instrument is setting the SRQ line. One function of a serial poll is to give the controller a way to find out which instrument is requesting service and clear the SRQ line. As part of the information in its serial poll response, each instrument must report whether or not it is requesting service. Any instrument that can request service must be able to respond to serial polls.

1.6.1 Serial Polls

The IBSPL routine conducts a serial poll of the instruments you specify. If more than one instrument is to be polled, the routine polls the instruments one after the other.

Each instrument polled has eight bits of status information in its interface; this group of bits is called the instrument's *status byte.* Each bit of the status byte is set (1) or clear (0) to report specific information about the instrument. When the computer serially polls the instrument, the instrument reports the state of each of these bits by setting or clearing the corresponding data line on the bus.

As shown in Figure 1–8, data line 6 contains the same type of information for all instruments. An instrument sets this line in response to a serial poll if it is requesting service from the computer. This line is different from the SRQ line, which is not one of the data lines. If an instrument has set the SRQ line, however, then when it is serially polled it must also set data line 6. Each instrument's designer determines the type of information that the instrument reports on each of the other data lines.

Figure 1–8: **Serial Poll Response**

Data Line	Meaning
7	Meaning is instrument-dependent
6	Set if the instrument is setting the SRQ line
5	Meaning is instrument-dependent
4	Meaning is instrument-dependent
3	Meaning is instrument-dependent
2	Meaning is instrument-dependent
1	Meaning is instrument-dependent
0	Meaning is instrument-dependent

MR-2123

1.6.2 Service Requests

Your program can detect a service request in either of two different ways. Which you choose will depend on how you wish to perform the detection operation itself and on how you wish subsequent serial polls to be initiated.

One of these ways is to use IBSRQ to designate a service subroutine. In this case, the IB device driver assumes the responsibility of detecting a subsequent service request and initating a serial poll. This procedure is identified as a "driver-initiated" serial poll.

A second way, which allows your program to retain the responsibility of detecting service requests, is to use IBSRQF to test the state of the SRQ line and, when the line is found to be asserted, to perform a serial poll using IBSPL or IBSTS. This procedure is identified as a "user-initiated" serial poll.

You can choose either the driver-initiated or user-initiated method of handling service requests through a call to IBSRQ. By specifying a service subroutine you indicate that you would like service requests to be handled by the driver. A call to IBSRQ with the service subroutine's name defaulted but with the delay flag set for delayed processing, leaves the responsibility of testing the SRQ line and performing serial polls with the user.

Driver-initated handling of service requests begins with detection of the SRQ–asserted condition. Once this occurs, the driver serially polls the list of devices capable of generating service requests as specified in a call to IBDEV. When a device is found to be requesting service, the service subroutine is called by the driver. This subroutine is user-written and has as its purpose the servicing of instruments requiring service.

When a service request is detected, your program receives the instrument address of the device which asserted the SRQ line. The variable which receives this information is used by your service subroutine to determine which instrument requested service and therefore the type of service to be rendered. You should place the variable which receives this information in a COMMON statement, because it is specified to the driver in your call to IBSRQ which is made in another part of your FORTRAN program.

An SRQ service subroutine has the same form as a normal subroutine, even though it is invoked by an instrument requesting service rather than by a CALL SUBROUTINE statement. The last statement executed in a service subroutine is a RETURN statement. With normal subroutines, the RETURN statement transfers control back to the statement following the CALL SUBROUTINE statement. With an SRQ service subroutine, the RETURN statement transfers control back to the statement following the last statement executed before the service subroutine was invoked. For more information about service subroutines, read the FORTRAN manual for your operating system.

The action required when an instrument requests service depends on the characteristics of the particular instrument. For example, one instrument on your IEEE bus might request service when it has new data; your program would ask it for the data by using the IBRECV routine. Another instrument might request service when it is out of paper; your program would type a warning on the terminal.

1.6.3 Parallel Polls

The IBPPL routine conducts a parallel poll. Every instrument that can do so responds to the poll. Each instrument polled has in its interface a bit of status information called the instrument's *status bit.* The meaning of each instrument's status bit is determined by the instrument's manufacturer and is described in its user's guide.

An instrument can respond to a parallel poll only if it is designed to respond and if its response has been *enabled.* Any instrument that can respond to a parallel poll must be able to have its reponse enabled either by local controls or by the computer, but not by both. The method of local poll enabling is not part of the IEEE standard, but is determined by the instrument's designer. The IBPPE routine enables the parallel poll response of instruments whose response can be enabled by the bus controller (the computer).

An instrument can respond to parallel polls until its parallel poll response is *disabled.* If the response was enabled locally, it is also disabled locally. If the parallel poll response was enabled by the IBPPE routine, however, it can be disabled by the IBPPD routine. The IBPPD routine affects selected instruments, while the IBPPU routine affects all instruments.

Enabling an instrument's parallel poll response assigns the instrument one of the bus's data lines and a condition. The instrument sets the data line during a parallel poll if the status bit in its interface is in the assigned condition. If the condition assigned to it is 0, the instrument sets the data line if its status bit is 0 at the time of the poll; if the condition assigned to it is 1, the instrument sets the data line if its status bit is 1 at the time of the poll.

For example, suppose your program enables instrument 17 with data line 5 and condition 0, and enables instrument 15 with data line 2 and condition 1. When your program conducts a parallel poll, the data lines have the values shown in Figure 1–9.

Figure 1–9: Example of a Parallel Poll Response

Data Line	Value
7	0
6	0
5	$\begin{cases} 0 \text{ if instrument 17's status bit is 1} \\ 1 \text{ if instrument 17's status bit is 0} \end{cases}$
4	0
3	0
2	$\begin{cases} 0 \text{ if instrument 15's status bit is 0} \\ 1 \text{ if instrument 15's status bit is 1} \end{cases}$
1	0
0	0

MR-2124

More than one instrument can be assigned to a single data line; this is not usually done, however, because if the line were set during a parallel poll, your program could not determine which instrument set it. Each of the eight data lines is clear in a parallel poll unless one or more instruments sets it.

NOTE

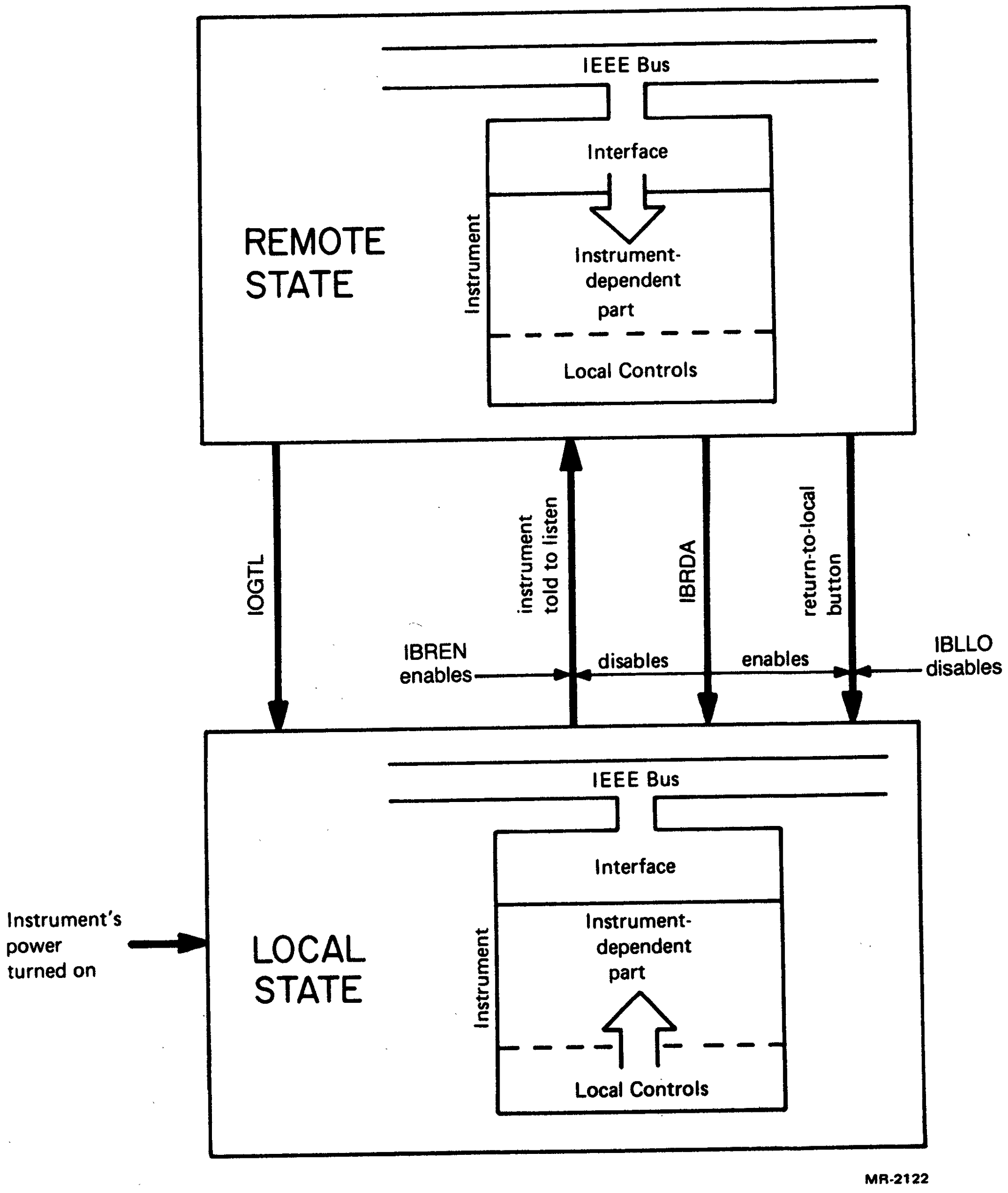
When more than eight instruments must be polled, two or more instruments may be assigned the same data line for their poll reply signal. If this is done, a logical one on the data line would signify that at least one of the asociated instruments had replied to the poll. A logical zero would signify that none of those instruments had replied.

1.7 Remote and Local

An instrument on the IEEE bus can use input information either from the bus or from manual controls on the instrument itself. When messages from the IEEE bus are the source of input information, the instrument is said to be in the remote state. When the manual controls on the instrument are the source of input information, the instrument is said to be in the local state. This section discusses how your program can control which state each instrument is in; these controls are summarized in Figure 1–10. A heavy arrow in this figure represents a transition between the local and remote states. A light arrow represents a statement that enables or disables the transition that the arrow points to.

No instrument can be in the remote state unless the remote enable (REN) bus line is set. When you start the computer, this line is clear, so all instruments on the bus are in the local state. The first IEEE bus routine to execute after the computer is started sets the REN line automatically. This line remains set while the computer communicates with instruments on the bus. An instrument enters the remote state when the computer tells it to listen while the REN bus line is set.

Figure 1–10: Remote and Local States



The IBRDA routine clears the REN bus line, causing all instruments on the bus to enter the local state. The IBGTL routine causes selected instruments to enter the local state, but it does not change the REN line. The IBREN routine sets the REN line, and the IBREMO routine reports whether the REN line is currently set or clear.

Some instruments have a return-to-local button, which can be used to put the instrument in the local state. The user's guide for a particular instrument tells you whether or not that instrument has a return-to-local button and, if so, where it is. The IBLLO routine disables the return-to-local buttons of all instruments on the bus and thus prevents each instrument from unexpectedly entering the local state (perhaps at a critical time) in the event that someone accidentally presses its return-to-local button. IBRDA still causes instruments to enter the local state even if IBLLO has been executed. IBREN cancels the effect of IBRDA, causing the return-to-local buttons to become operative again.

The instrument designer determines how the instrument behaves under local control and under remote control. When going from local to remote control, an instrument can either use its current local settings until they are subsequently overridden by remote input or use remote input that was previously received. In either case, the instrument must ignore future use of its local controls and become responsive to remote input. Some instruments, however, have functions that are always controlled locally, even in the remote state. When going from remote to local control, an instrument can either use input from its local controls immediately or continue to use the last input from the bus until that input is overridden by subsequent local control settings. In either case, the instrument must ignore future remote input and respond to future use of its local controls. It can still talk and listen while in the local state.

1.8 Resets

Your program can separately clear, or reset, either the instrument's interface to the bus or its instrument-dependent part.

1.8.1 Clearing Interfaces

The IBIFC routine clears the bus and every instrument's interface to the bus by setting the interface clear (IFC) bus line, a line reserved for this purpose by the standard. Note that the IBIFC routine does not clear the SRQ line. This routine clears only the interfaces, not the instrument-dependent parts of instruments. Each interface returns to the clear state defined by the IEEE standard. This routine has the following effects:

1. All return-to-local buttons of bus instruments become operative. If IBLLO has been called to disable these buttons, it is no longer in effect.
2. The REN bus line is cleared and then set.

3. All instruments enter the local state. However, because the REN bus line is set, each instrument enters the remote state when it is told to listen.

4. Any condition set by a message (for example, a range or a sampling rate set by an IBSEND routine) is not affected by this routine.

You can use IBIFC at the beginning of your program to undo any effect that previous IEEE bus routines have had on the interfaces.

1.8.2 Clearing Instruments

Two routines clear the instrument-dependent parts of instruments on the bus. The IBDCL routine clears all bus instruments, and the IBSDC routine clears only selected instruments. These routines do not clear the instrument's interfaces. Each instrument cleared returns to a clear state defined by that instrument's manufacturer; this is usually the state the instrument is in after its power is turned on. Refer to the user's guide for the particular instrument for the properties of this state.

You can use the IBDCL routine at the beginning of your program to undo the effects of previous message routines.

CAUTION

Never turn on an instrument's power while the computer is running!

1.9 Asynchronous Processing

A program written in FORTRAN IV normally executes in a sequential manner. That is, the process initiated by one statement of the program completes before the process initiated by the next statement begins.

Such sequential processing by the computer can be interrupted, however, by signals sent to the computer from peripheral hardware devices. These signals not only interrupt the normal sequential execution of program statements. They also direct the computer to execute an alternate set of instructions designed to enable communication between the computer and the interrupting peripheral device. These alternate instruction sets are referred to as Interrupt Service Routines (ISRs).

In most cases, an ISR "saves" (keeps track of) the state of the computer at the time it is interrupted. Then, after the ISR has finished using the computer to execute its own set of instructions, it restores the computer to that prior state. The computer is then free to continue processing the statements of the main program.

This capability of interrupting main programs to execute an alternate set of instructions is called asynchronous processing. An ISR enables the computer to process two sets of instructions: those in the main program and the alternate set provided in the ISR. Although these operations do not use the computer at precisely the same time, processing is so fast that the operations seem to occur simultaneously.

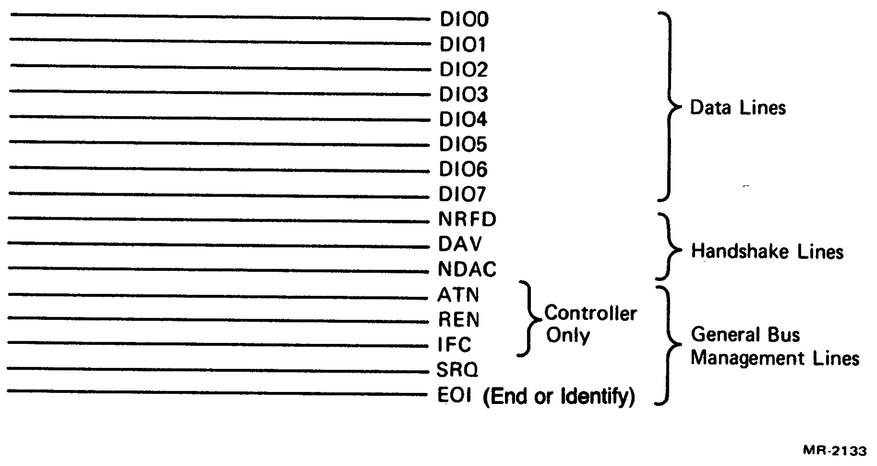
Since an asynchronous process can be invoked at anytime, it is important to be very careful when the interactive process between the main routine and the Interrrupt Service Routine ISR are designed. The (ISR) should not change a value that the main program expects to be unchanged.

Chapter 2
Advanced Concepts

Until now we have emphasized what the IEEE bus does, not how it works. This chapter discusses in detail how the IEEE bus and the IEEE routines work. For a few applications you do not need the information here. This chapter is for people who need to know how the IEEE routines control the bus.

As you know from Chapter 1, the bus has 16 lines: eight data lines, three handshake lines, and five general bus management lines. These are illustrated in Figure 2–1. Each line is set when it is grounded, so that a bus line is set if one or more instruments set it. A line is clear only if no instrument sets it. The state of the line is the logical OR of all the instruments that assert it.

Figure 2–1: The IEEE Bus Lines



2.1 Commands: The Attention (ATN) Line

The computer is the controller and controls the bus by sending *commands* to the interfaces. A command is a character the computer sends on the bus while the attention (ATN) bus line is set. Only the computer can send a command or change the ATN line. A command differs from a message in the following ways:

1. A character is a command if it is sent when ATN is set and is part of a message if it is sent when ATN is clear.
2. A command is sent by the controller (the computer), a message by a talker.
3. A command is received by all instruments, a message only by listeners.
4. A command is a directive to the instrument's interface, so the interpretation of a command is defined by the standard. A message, however, is sent through the interface to the instrument; its interpretation depends on the instrument's vocabulary, which is defined by the instrument's designer.

2.1.1 Instrument Addressing

Some commands tell a certain instrument to talk or listen. When your program specifies a talker address in an IEEE routine, the computer directs the instrument with that address to talk by sending the appropriate MTA (my talk address) command, in the range MTA0 to MTA30. An instrument becomes the talker when its interface detects the MTA command for its address. The instrument can send message characters when the computer clears the ATN line. It stops being the talker when it detects any other MTA command; this assures that at most one instrument is a talker at any one time. The instrument also stops being the talker when it detects the UNT command. The UNT command is sent by the IBUNT routine or any other routine in which the computer is the talker so that no instrument talks when the computer clears the ATN line.

For example, instrument 15 becomes the talker when the computer sends the MTA15 command. It stops being the talker when the computer sends MTA0 to MTA14, MTA16 to MTA30, IBUNT, or IBIFC.

In a similar way, when your program specifies a listener address in an IEEE routine, the computer directs the instrument with that address to listen by sending the appropriate MLA (my listen address) command, in the range MLA0 to MLA30. An instrument becomes a listener when its interface detects the MLA command for its address. The instrument can receive message characters when the computer clears the ATN line. It does not stop being a listener when it detects any other MLA command; this allows more than one instrument to be a listener at the same time. An instrument stops being a listener only when the computer sends the UNL command, after which no instrument is a listener. This is done by the IBUNL routine every time a new listener list is specified.

When your program specifies a secondary address, the computer sends the appropriate MSA (my secondary address) command, in the range MSA0 to MSA30.

2.1.2 Universal and Addressed Commands

The other commands the computer sends are classified as either universal commands or addressed commands. *Universal commands* affect all instruments, while *addressed commands* affect only the addressed instruments. The names of these commands are listed in Figure 2–4 and Appendix B.

2.1.3 Control Conflict

Only the computer, the bus controller, can set or clear the ATN, IFC, and REN bus lines. If another controller changes any of these three lines, the computer produces an error message ("conflict over control of the bus", error number 14; see Chapter 9) when the next IEEE routine is invoked. The computer also produces this error message if the total cable length of your IEEE bus is too long or if there is an impedance problem because your instrument does not meet the IEEE standard.

2.2 The End or Identify (EOI) Line

The EOI (End Or Identify) bus line has two functions. If the ATN line is clear, the EOI line is used by the talker as the End-of-Message (EOI) line to mark the last character of the message. If the ATN line is set, the EOI line is used by the controller as the IDY (identify) line to conduct a parallel poll.

2.2.1 Parallel Polls: Identify (IDY)

The computer conducts a parallel poll by setting both the ATN and IDY lines, waiting at least two microseconds, and then reading the bus's data lines.

2.2.2 Fragmented Messages EOI

In Chapter 1 we pointed out that the IBSEOI routine sets the EOI line while sending the last character of the string to mark that character as the last one of the message. The IBSEND routine is identical to IBSEOI except that it does not set the EOI line while sending the last character.

A call to IBSEND, IBSEOI, or IBASND can specify the number of characters to be sent in one of three ways. If the message length argument is defaulted or equal to –1, the contents of the array are sent until a null byte (octal 0) is reached. If the message length is equal to zero, no message is sent. And finally, if the message length is a positive integer greater than zero, it specifies the number of characters to be sent. In this case, octal 0 is sent like any other character.

If a very long message is to be sent, null byte termination is the simplest method because it does not require you to specify the exact length of the message, and because if the length is greater than 65534 characters, only one call to the IBS software will handle the entire transaction. Should the message itself contain 0 bytes, the actual message length must be specified as a positive integer greater than zero but less than or equal to 65534 (see Appendix F for information about handling integers larger than 32,767).

Therefore, if a counted message contains 65535 charcters or more, it must be sent in fragments, with each message fragment containing up to 65534 characters, and using the EOI line to signal the end of the message. The EOI line should be set only at the end of the entire message, not at the end of each fragment of the message. To do this, use IBSEND for every message fragment except the last, and use IBSEOI for the last fragment of the message.

2.3 The Data Lines

The eight data lines are used for messages, commands, and status information. They are called DIO0 to DIO7 (Data In/Out). The IEEE standard numbers the lines 1 to 8, but in this book we number them 0 to 7 for consistency with the computer. The data lines can represent numbers in the range 0 to 255. Each data line is associated with a value, as shown in Figure 2–2. The number represented by the data lines is the sum of the values associated with the lines that are set. For example, if lines 0, 1, 2, and 5 are set, the data lines represent the number 39 (= 1 + 2 + 4 + 32).

Figure 2–2: Values Associated with the Data Lines

Data Line	Associated Value
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

MR-2125

2.3.1 Message Codes

The data on the data lines can be binary data or ASCII characters. Each ASCII character is associated with a unique number in the range 0 to 127, called its ASCII value. The binary data values are defined by the user in the instrument user's manual. A talker sends a message character by setting the data lines so that they represent the ASCII value of that character. Figure 2–3 lists the characters and their representation on the data lines as binary numbers. For example, the character "A" is represented by data lines 0 and 6 set. Appendix A also lists the ASCII values and their octal, decimal, and hexadecimal equivalents.

To send a nonprinting character as part of a message, use the function described in the Programmer's Reference manuals for your operating system. Use the character's ASCII value and concatenate that string with the printing characters in the message.

To make a character a message terminator, specify its ASCII value in the argument of the IBTERM routine.

Figure 2–3: ASCII Character Codes

Data Line # 7 6 5 4 / 3 2 1 0	0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1
0 0 0 0	NUL	DLE	SP	0	@	P	'	p
0 0 0 1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	STX	DC2	"	2	B	R	b	r
0 0 1 1	ETX	DC3	#	3	C	S	c	s
0 1 0 0	EOT	DC4	$	4	D	T	d	t
0 1 0 1	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	ACK	SYN	&	6	F	V	f	v
0 1 1 1	BEL	ETB	'	7	G	W	g	w
1 0 0 0	BS	CAN	(	8	H	X	h	x
1 0 0 1	HT	EM	)	9	I	Y	i	y
1 0 1 0	LF	SUB	*	:	J	Z	j	z
1 0 1 1	VT	ESC	+	;	K	[	k	{
1 1 0 0	FF	FS	,	<	L	\	l	¦
1 1 0 1	CR	GS	-	=	M	]	m	}
1 1 1 0	SO	RS	.	>	N	^	n	~
1 1 1 1	SI	US	/	?	O	_____	o	DEL

MR-2135

2.3.2 Command Codes

Each command has a numeric code and therefore a representation on the data lines. These representations are shown in Figure 2–4. This figure shows all of the command codes supported by the IEEE standard. Figure 2–5 shows the format of these command codes. Many of the possible codes for addressed and universal commands have not yet been assigned meanings by the standard. Chapter 5 explains how to address and send messages to instruments on the IEEE Bus using IB routines.

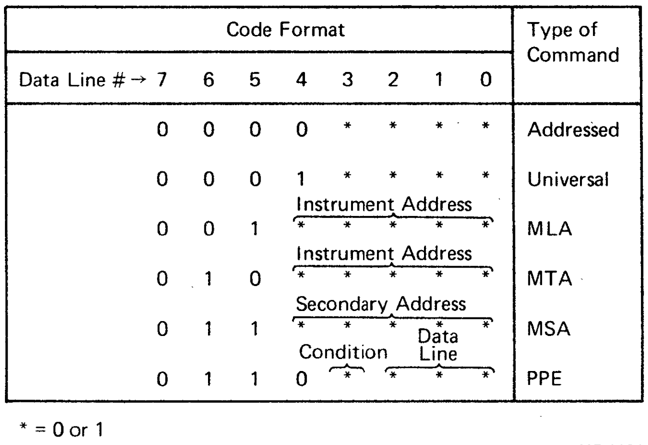
Figure 2–4: Command Codes

	Data Line # 7 6 5 4 3 2 1 0	Command Abbreviation	Command Name
Addressed Commands	0 0 0 0 0 0 0 1	GTL	Go To Local
	0 0 0 0 0 1 0 0	SDC	Selected Device Clear
	0 0 0 0 0 1 0 1	PPC	Parallel Poll Configure
	0 0 0 0 1 0 0 0	GET	Group Execute Trigger
Universal Commands	0 0 0 1 0 0 0 1	LLO	Local Lockout
	0 0 0 1 0 1 0 0	DCL	Device Clear
	0 0 0 1 0 1 0 1	PPU	Parallel Poll Unconfigure
	0 0 0 1 1 0 0 0	SPE	Serial Poll Enable
	0 0 0 1 1 0 0 1	SPD	Serial Poll Disable
Listener Address Commands	0 0 1 0 0 0 0 0	MLA0	My Listen Address 0
	0 0 1 0 0 0 0 1	MLA1	My Listen Address 1
	⋮	⋮	⋮
	0 0 1 1 1 1 1 0	MLA30	My Listen Address 30
	0 0 1 1 1 1 1 1	UNL	Unlisten
Talker Address Commands	0 1 0 0 0 0 0 0	MTA0	My Talk Address 0
	0 1 0 0 0 0 0 1	MTA1	My Talk Address 1
	⋮	⋮	⋮
	0 1 0 1 1 1 1 0	MTA30	My Talk Address 30
	0 1 0 1 1 1 1 1	UNT	Untalk
Secondary Address Commands	0 1 1 0 0 0 0 0	MSA0	My Secondary Address 0
	0 1 1 0 0 0 0 1	MSA1	My Secondary Address 1
	⋮	⋮	⋮
	0 1 1 1 1 1 1 0	MSA30	My Secondary Address 30
Parallel Poll Enable Commands			Parallel Poll Enable
	0 1 1 0 0 0 0 0	PPE	Condition 0, Data Line 0
	0 1 1 0 0 0 0 1	PPE	Condition 0, Data Line 1
	⋮	⋮	⋮
	0 1 1 0 0 1 1 1	PPE	Condition 0, Data Line 7
	0 1 1 0 1 0 0 0	PPE	Condition 1, Data Line 0
	0 1 1 0 1 0 0 1	PPE	Condition 1, Data Line 1
	⋮	⋮	⋮
	0 1 1 0 1 1 1 1	PPE	Condition 1, Data Line 7
	0 1 1 1 0 0 0 0	PPD	Parallel Poll Disable

MR-2136

Notice that the codes for secondary address commands and parallel poll enable commands overlap. Command codes in this range are interpreted as secondary address commands if they follow a talker or listener address command; they are interpreted as parallel poll enable commands if they follow a parallel poll configure command, which is one of the addressed commands.

Figure 2–5: Command Code Format



Code Format									Type of Command
Data Line # →	7	6	5	4	3	2	1	0	
	0	0	0	0	*	*	*	*	Addressed
	0	0	0	1	*	*	*	*	Universal
				Instrument Address					
	0	0	1	*	*	*	*	*	MLA
				Instrument Address					
	0	1	0	*	*	*	*	*	MTA
				Secondary Address					
	0	1	1	*	*	*	*	*	MSA
					Condition		Data Line		
	0	1	1	0	*	*	*	*	PPE

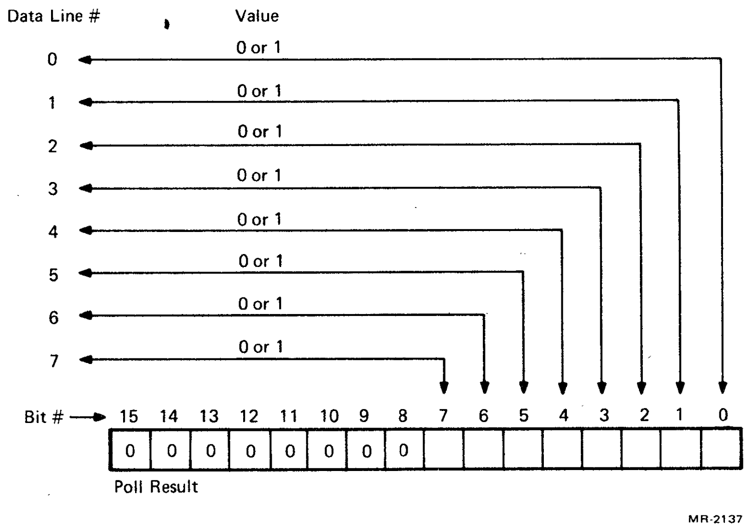
* = 0 or 1

MR-2134

2.3.3 Poll Results

Both parallel and serial poll results are received on the data lines and are therefore numbers between 0 and 255. Often you need to know whether a certain data line was set or clear. Bits 0 through 7 of the poll result argument correspond to the data lines 0 through 7 during the poll, as shown in Figure 2–6.

Figure 2–6: Poll Result Bits Correspond to Data Lines



2.4 The Handshake Lines

2.4.1 The Handshake

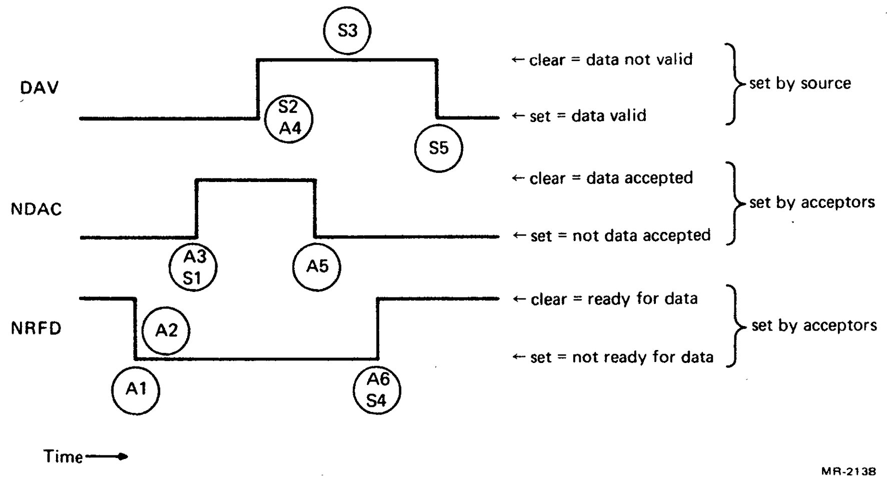
Three bus lines provide a *handshake* mechanism to ensure that every character sent is received. This handshake must have both "hands": for message characters, the *source handshake* is provided by the talker and the *acceptor handshake* is provided by the listeners; for command characters, the source handshake is provided by the computer and the acceptor handshake is provided by the *command acceptors*, which are all the instruments on the bus. Because there is a handshake for every character, the speed of the transmission is limited by the slowest instrument involved in the transmission.

The DAV (data valid) line indicates whether or not a character is available and valid on the data lines. The source sets this line when the character on the data lines is valid.

The NRFD (not ready for data) line indicates whether or not the acceptors are ready for the next character of data. Each acceptor sets this line if it is not ready for data. The line is clear only if no acceptor is setting it, which means that every acceptor is ready for the next character.

The NDAC (not data accepted) line indicates whether or not the acceptors have accepted the current character of data. Each acceptor sets this line if it has not accepted the current character. The line is clear only if no acceptor is setting it, which means that every acceptor has accepted the current character.

Figure 2–7: A Handshake



A detailed description of the handshake mechanism is available in the IEEE standard. The following brief description starts at the time when one character is on the bus and DAV has been set by the source to indicate that the character is valid data. Figure 2–7 shows the state of the three handshake lines during one handshake. Time is from left to right in the figure but is not to scale.

The source:

S1. Waits for NDAC to be cleared, which indicates that all acceptors have accepted the character of data.

S2. Clears DAV, indicating to the acceptors that the data lines no longer contain valid data.

S3. Changes the data lines to the next character.

S4. Waits for NRFD to be cleared, which indicates that all acceptors are ready for the next character of data.

S5. Sets DAV, indicating to the acceptors that the next character is on the data lines.

Each acceptor:

A1. Sets NRFD, indicating to the source that it is not ready for the next character of data. This line sets when the first acceptor sets it.

A2. Reads the current character.

A3. Stops setting NDAC. This line remains set until the last acceptor stops setting it, at which time it becomes clear, indicating to the source that all acceptors accepted the current character of data.

A4. Waits for the source to clear the DAV line.

A5. Sets NDAC to prepare for the next handshake. This line sets when the first acceptor sets it.

A6. Stops setting NRFD when it is ready to accept the next character. This line remains set until the last acceptor stops setting it, at which time it clears, indicating to the talker that all acceptors are ready for the next character of data.

2.4.2 Error Conditions

If the source tries to send a character and there is no acceptor, NRFD and NDAC are both clear, a condition that never occurs when there is an acceptor handshake. This causes the computer to produce a no valid listener on the bus error (see Chapter 9).

This error condition can be caused by one of the following bus conditions:

1. There is no instrument on the bus, hence no command acceptor.
2. There is no instrument on the bus with the address of the listener specified.
3. The instrument specified as a listener cannot listen, is set locally to "talk only," or is turned off.

The IBLNR routine tests for this error condition by trying to send a one-character message (a line feed) to the instruments specified. This routine does not produce an error if it successfully addresses a valid listener, but does not complete the message handshake. Instead, it reports the error to your program in its argument. This routine does produce an error, however, if there is no instrument on the bus, hence no command acceptor.

The absence of a talker does not produce an error message. The following conditions result in the absence of a talker:

1. There is no instrument on the bus with the address of the talker you specified.
2. The instrument specified as a talker cannot talk, is set locally to "listen only," or is turned off.
3. The instrument is not designed to respond to a serial poll (IBSRQ, IBSTS, and IBSPL routines only).
4. Impedance problems caused by an improper configuration or by an instrument that does not comply with the IEEE specification.

   You can use the IBTIMO routine to set a time limit for each handshake, so that the computer generates an error instead of waiting indefinitely for the next character from the talker when one of the above conditions occurs.

2.4.3 The Idle State

At the end of every IEEE routine, the computer puts the bus in an idle state. The computer sets the NRFD line to prevent the talker from sending any characters. The ATN line is asserted, except at the end of the IBFREE routine.

Chapter 3
System Hardware Installation

The IBV11–A can be installed in any LSI–11 system configuration. It can be included among user-assembled component systems, and be added to the PDP–11/03 packaged LSI–11 microcomputer system or to the PDP–11/03 floppy disk system.

This chapter explains how to install the IBV11–A interface to the IEEE standard instrument bus in an LSI–11 system. See the *IB11 UNIBUS to IEEE Instrument Bus Interface Installation Manual* for information on how to install the IBV11–A interface in other PDP–11 systems.

3.1 Installing the IBV11–A Instrument Bus Interface Module

You can expand your LSI–11 system to a programmable instrument system by installing the IBV11–A (see Figure 3–1) in the console backplane. The IBV11–A has two switch packs, S1 and S2 (see Figures 3–2 and 3–3) to set the device address and vector address. These switch packs for the first IBV11–A unit should be set correctly at the factory for your installation.

The standard address configuration for MINC and DECLAB–11/MNC users is 171420 (octal) as the Control Status Register (CSR) address (S2), and 420 (octal) as the vector address (S1) for the first IBV11–A unit. For all other PDP–11 users, the standard CSR address is 160150 (octal). Note that only the IB status register address is configured by using switch pack two. The IB data register address is always the IB status register address plus 2. Similarly, only the error interrupt vector address is configured by using switch pack one. The remaining three vector addresses are permanently assigned sequential addresses in address increments of 4 (octal).

If you have more than one IBV11–A unit, each unit must have a unique address setting. Successive units should be assigned sequential CSR addresses (switch pack two) in address increments of 10 (octal). Vector addresses should also be assigned sequentially (switch pack one) though in address increments of 20 (octal). If an IBV11–A unit is installed with its switches set differently from the standard ones, or if you are using multiple units, see Chapter 4 of this manual. Multiple unit settings must be set according to the address increments described above for proper operation with the IB software.

Example:

```
on a MINC system:
IBCSR0 (the CSR address for unit 0) = 171420
IBVEC0 (the base vector address for unit 0) = 420
IBCSR1 = 171430
IBVEC1 = 440
etc.

on a non-MINC system:
IBCSR0 = 160150
IBVEC0 = 420
IBCSR1 = 160160
IBVEC1 = 440
etc.
```

With the computer system's power turned off, insert the IBV11–A into the console backplane. There must be no empty positions between the first and last filled sequential positions in the backplane. (See the *Microcomputer Processor Handbook.*) With the IBV11–A oriented the same as the LSI–11 modules, align the IBV11–A with the guides on the side of the module case, and push it toward the rear until you feel some resistance. Then push more firmly, see-sawing the module as you seat its connector fingers in the backplane.

Next, attach the BN11A connector cable to the IBV11–A. One end of the BN11A cable has a single 20–pin connector that mates with a complementary connector on the module. The other end has a two-sided 24–pin connector. Pass the end with the 20–pin connector through an access slot in the rear panel of the console cabinet. Orient the cable to the module so that the extreme end of the connector is toward the latch on the module connector (see Figure 3–1). Apply pressure to the cable connector until the module latch snaps over the connector.

Figure 3–1: IBV11–A Module and BN11A Connector Cable

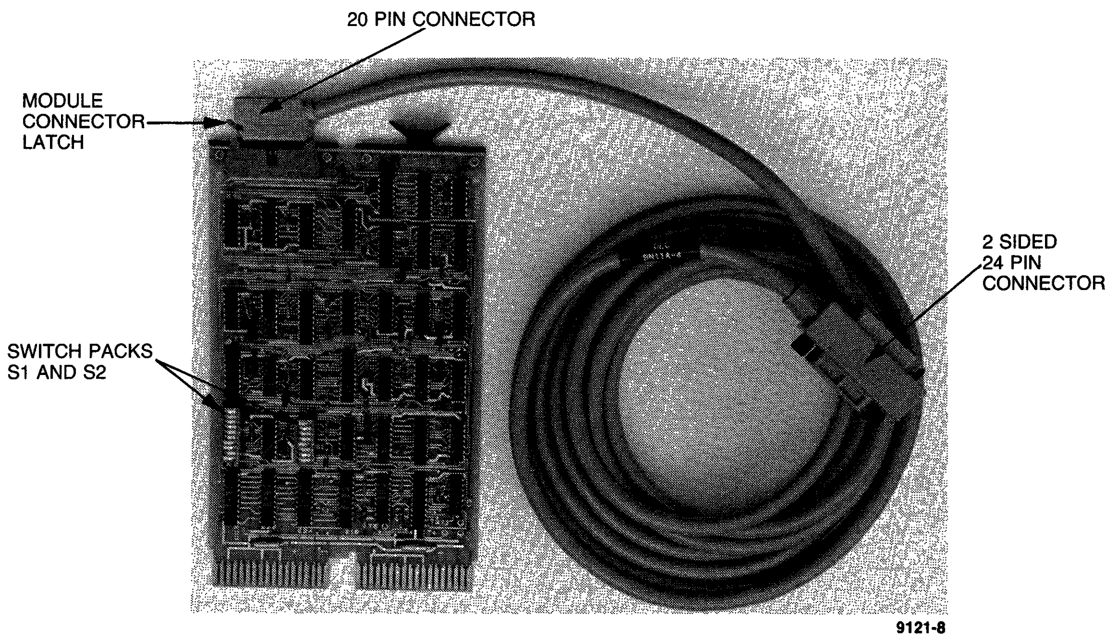


Figure 3–2: S1 and S2 Switch Pack Settings for a MINC or DECLAB–11/MNC

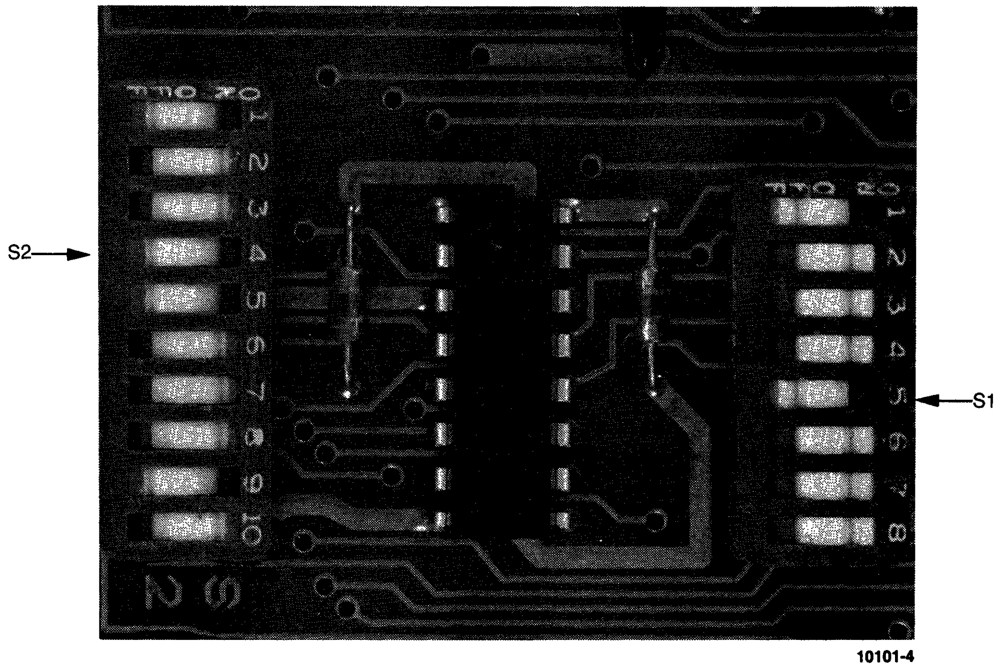
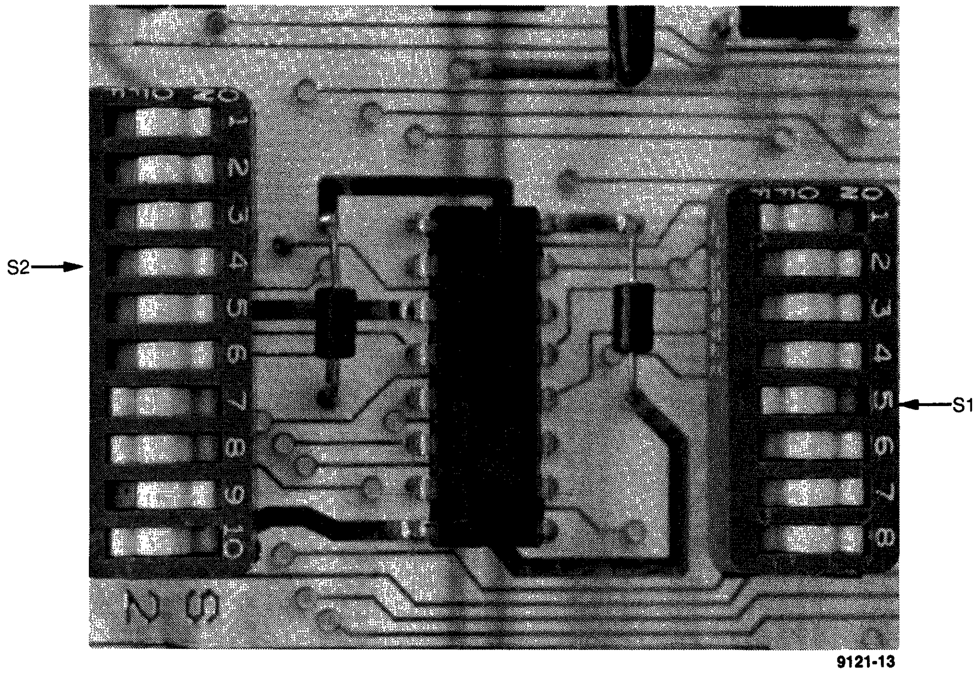


Figure 3–3: **S1 and S2 Switch Pack Settings for a PDP–11 other than a MINC or DECLAB–11/MNC**



3.2 Instrument Addresses

Before you can operate your instrument system under the control of the IBV11–A/IB11 instrument bus interface, you must assign an address to each instrument. This instrument address is a number that identifies each instrument.

When you want an instrument to receive data or programmed instruction, you enter its instrument address in a listener specification of an IB routine. The FORTRAN IB routines automatically translate this into its corresponding ASCII character, in the range of proper listener addresses conforming to the IEEE standard (see Appendix B). The corresponding octal code is sent on the instrument bus, and the instrument is ready to receive data or programmed instruction.

Similarly, an instrument address given in a talker specification is automatically translated by the FORTRAN IB routines into its corresponding ASCII character in a range of proper talker addresses within the IEEE standard. When the corresponding octal code is sent on the instrument bus, the instrument with that address assigned is ready to send data.

3.2.1 Setting Instrument Addresses — Primary Addresses

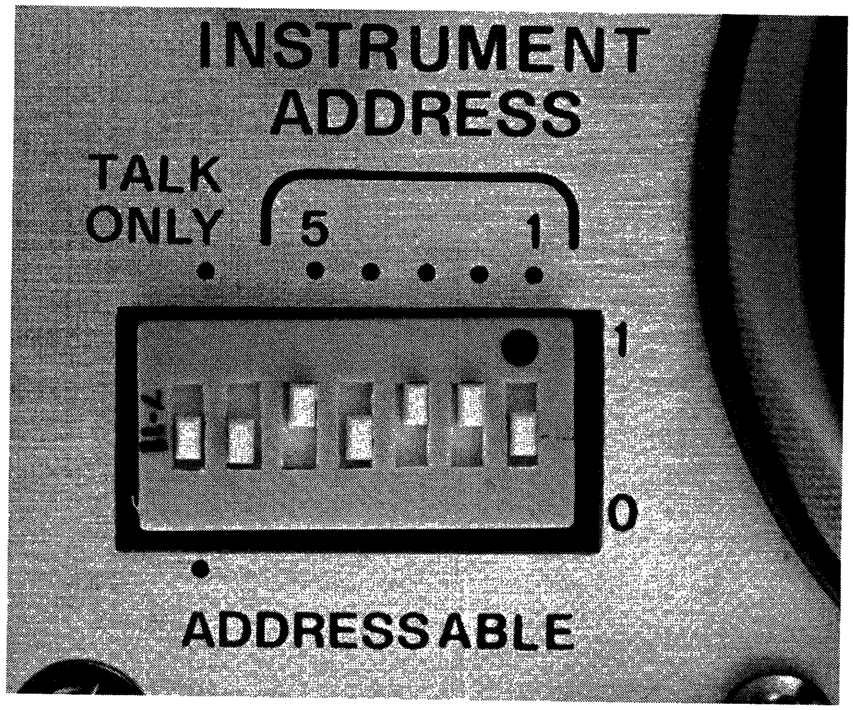
Most instruments that connect to the instrument bus have sets of switches resembling those in Figure 3–4. Each switch in an instrument address switch set represents one position in binary code. You set the switches to correspond to the binary equivalent of the assigned decimal instrument address. A switch set to ON usually assigns a 1; a switch set to OFF usually assigns a 0. For example, if you assign the address 6 to an instrument, its corresponding binary code switch settings will be as follows:

b(5)	b(4)	b(3)	b(2)	b(1)	
0	0	1	1	0	= 6

There are other ways to set instrument addresses. For example, some instruments use pairs of jumper posts to represent binary code. Usually, when a jumper is connected across a pair of posts, a 1 is assigned to that position in the binary code. An unconnected pair of jumper posts usually assigns a 0.

The operation manual for each instrument describes the method used, and how to set the assigned instrument address. The operation manual also tells whether the instrument has its instrument address preset, and whether it can be changed. There are 31 instrument addresses recognized by the IB routines; therefore it should not be necessary to change any of the preset instrument addresses unless instruments with distinct operations have identical preset instrument addresses.

Figure 3–4: Typical Instrument Address Switches



The 31 instrument addresses (0 through 30) and their corresponding binary, octal, and hexadecimal switch settings are listed in Table 3–1.

Table 3–1: Instrument Addresses and Binary Address Switch Settings

Instrument Address (Decimal)	Binary Address Switch Settings (1 = switch on) A(5)	A(4)	A(3)	A(2)	A(1)	Octal Address	Hexadecimal Address
0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1
2	0	0	0	1	0	2	2
3	0	0	0	1	1	3	3
4	0	0	1	0	0	4	4
5	0	0	1	0	1	5	5
6	0	0	1	1	0	6	6
7	0	0	1	1	1	7	7
8	0	1	0	0	0	10	8
9	0	1	0	0	1	11	9
10	0	1	0	1	0	12	A
11	0	1	0	1	1	13	B
12	0	1	1	0	0	14	C
13	0	1	1	0	1	15	D
14	0	1	1	1	0	16	E
15	0	1	1	1	1	17	F
16	1	0	0	0	0	20	10
17	1	0	0	0	1	21	11
18	1	0	0	1	0	22	12
19	1	0	0	1	1	23	13
20	1	0	1	0	0	24	14
21	1	0	1	0	1	25	15
22	1	0	1	1	0	26	16
23	1	0	1	1	1	27	17
24	1	1	0	0	0	30	18
25	1	1	0	0	1	31	19
26	1	1	0	1	0	32	1A
27	1	1	0	1	1	33	1B
28	1	1	1	0	0	34	1C
29	1	1	1	0	1	35	1D
30	1	1	1	1	0	36	1E

Use these addresses and no others. Regardless of the method used to make the binary settings, each instrument must be assigned a unique address by which it is recognized. The instrument address is also called the primary address.

3.2.2 Instruments with Alternate Functions — Secondary Addresses

Some instruments can function as talker or listener in more than one way. These functions are distinguished by a set of 31 secondary addresses in the range 0 through 30. These addresses are predetermined by instrument manufacturers and usually cannot be changed. Their use in address specifications is described in Section 5.2.1.

3.2.3 Addressable Mode

Many instruments have a switch that lets you select either addressable or independent operating modes. For programmed operation under control of the IBV11–A/IB11 Instrument Bus Interface, set this switch to the addressable position.

3.2.4 Duplicate Addresses

In general, two instruments should not have the same primary address. To avoid assigning duplicate addresses, it is good practice to tabulate the addresses assigned to all instruments. List the instrument by name with its primary address, as well as any secondary addresses. This list also makes a useful reference for the programmer to specify instruments as talkers or listeners during a programming session. Appendix D contains a sample instrument address form that you can use for this purpose.

3.2.5 Checking Instrument Addresses

After you have assigned addresses to all instruments, ensure that the following statements are valid for your system:

- Each instrument has its own unique address, and a written record showing the assignment of this address set to an instrument has been made in a form suitable for easy reference. Only primary addresses in the range 0 through 30 have been used. Any secondary addresses (in the range 0 through 30) have been recorded.
- Each instrument with an addressable control switch has been set to the addressable mode so that it can operate under control of the IBV11–A/IB11.

3.2.6 How to Determine What your Instrument Addresses Are

Some IEEE instrument manuals refer to instrument addresses in a way that differs from the IB addressing scheme. However, the names used for the two types of addresses are similar. A manufacturer may suggest that you set the address switch of an instrument to a particular "talk address" or "listen address," while argument lists for some IB routines, such as IBSEND and IBRECV include talkers and listeners. The address of a talker or listener in FORTRAN is not quite the same as the manufacturer's instructions. The IB software requires you to address an instrument by a single decimal number, which differs from, but is related to, the manufacturer's "talk" and "listen" addresses. This section explains how you can derive this decimal address from addresses given by a manufacturer and how the IB software uses the address.

Let's look at an instrument whose manufacturer's instructions advise you to set the "talk address" to V and the "listen address" to 6. First, "V" and "6" are ASCII characters and ASCII characters are one of many compact notations for binary numbers. The binary numbers might be represented in octal, decimal, hexadecimal or ASCII as well as in binary. To determine your instrument's IB address, do the following:

- In the Master IEEE Bus Address Table in Figure 3–5, find the row that contains ASCII "talk address" V and ASCII "listen address" 6.
- Locate the instrument's IB address in the left column of that row. The number in this column is 22. This is the instrument address you must use in IB routines to refer to this instrument either as a talker or a listener.

NOTE

> An IB primary address of zero must be specified using its ASCII "talk address" or ASCII "listen address." An IB primary address of zero, if specified as a value of zero, will be interpreted by the IB routines as terminating the address list.

To understand how the IB address is derived, examine some of the other numbers in the row. Note that the binary instrument address is 10110. This number equals the last 5 bits of the binary representation of both the "talk address" and the "listen address." The decimal instrument address in the table is the decimal equivalent of the binary instrument address. The decimal instrument address in the row we've been using is 22. This number, in turn, equals the IB address. You must also set the address switch on your instrument. The most common types of address switches are binary and hexadecimal. If you do not know which type your instrument has, consult your instrument manual. Set the switch to the number in the instrument address column that corresponds to your type of switch. If the switch is binary, use the binary instrument address from the table. If the switch is hexadecimal use the hexadecimal instrument address.

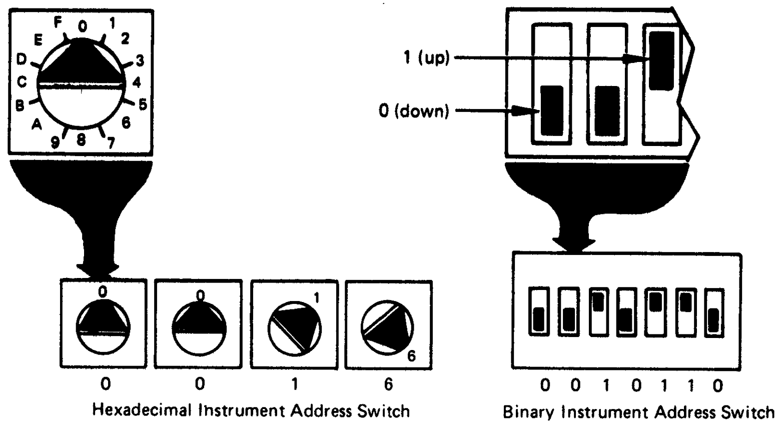
3.3 Bus Cables

After you have correctly set all instrument address switches, you must connect the instruments to the IBV11–A. The BN11A cable attached to the IBV11–A (see Section 3.1), has on its free end a two-sided 24–pin connector. One side is a male 24–pin connector and the other side is a female 24–pin connector. The BN11A connector conforms to the IEEE 488–1978 standard and can be connected to any instrument that also conforms to that standard. Attach the connector to an instrument in your system.

3.3.1 Connecting Bus Cables

You can connect the remaining instruments in your system to the IBV11–A with BN01A bus cables. These cables have the IEEE standard two-sided 24–pin connectors on both ends and connect to all IEEE bus-compatible instruments, as well as to each other. They are available in lengths of 1, 2, and 4 meters (3.3, 6.6, and 13.1 feet).

Figure 3–5: Instrument Address Switch Information


F 0 1 2 3 4 5 6 7 8 9 A B C D E
1 (up)
0 (down)
0
0
1
6
0 0 1 0 1 1 0
Hexadecimal Instrument Address Switch
Binary Instrument Address Switch

Note: Instrument address switches are usually found on the rear panel of an instrument.

MASTER IEEE BUS ADDRESS TABLE

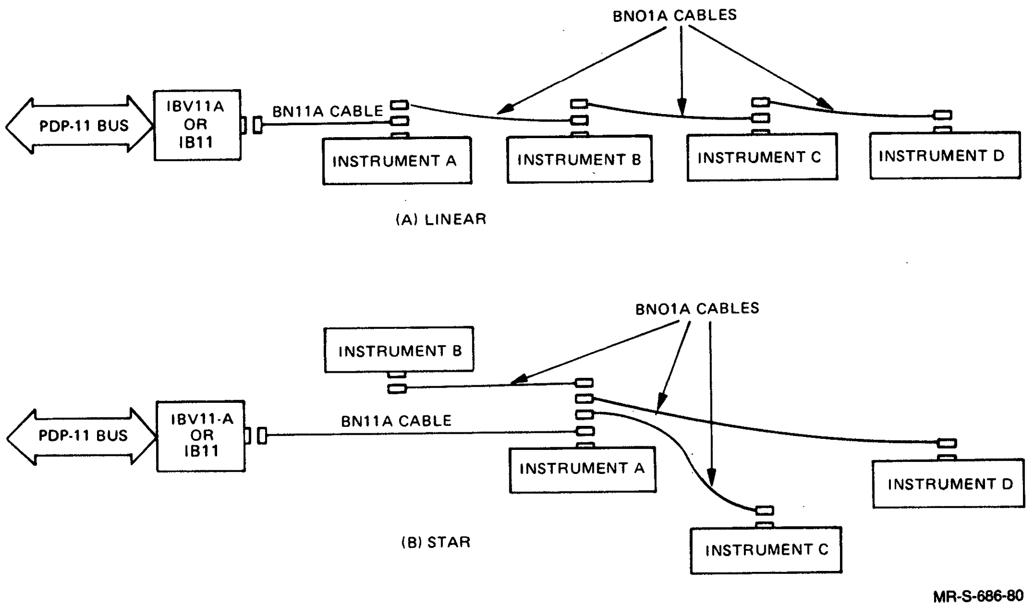
Instrument Addresses	"talk address" MTA		"listen address" MLA		Instrument Address (the 5 low-order bits of the talk and listen addresses) Set your instrument's address switch to one of these			
	ASCII	Octal	ASCII	Octal	Octal	Decimal	Hexadecimal	Binary
0	@	100	space	40	0	0	0	00 000
1	A	101	!	41	1	1	1	00 001
2	B	102	"	42	2	2	2	00 010
3	C	103	#	43	3	3	3	00 011
4	D	104	$	44	4	4	4	00 100
5	E	105	%	45	5	5	5	00 101
6	F	106	&	46	6	6	6	00 110
7	G	107	'	47	7	7	7	00 111
8	H	110	(	50	10	8	8	01 000
9	I	111	)	51	11	9	9	01 001
10	J	112	*	52	12	10	A	01 010
11	K	113	+	53	13	11	B	01 011
12	L	114	,	54	14	12	C	01 100
13	M	115	-	55	15	13	D	01 101
14	N	116	.	56	16	14	E	01 110
15	O	117	/	57	17	15	F	01 111
16	P	120	0	60	20	16	10	10 000
17	Q	121	1	61	21	17	11	10 001
18	R	122	2	62	22	18	12	10 010
19	S	123	3	63	23	19	13	10 011
20	T	124	4	64	24	20	14	10 100
21	U	125	5	65	25	21	15	10 101
22	V	126	6	66	26	22	16	10 110
23	W	127	7	67	27	23	17	10 111
24	X	130	8	70	30	24	18	11 000
25	Y	131	9	71	31	25	19	11 001
26	Z	132	:	72	32	26	1A	11 010
27	[	133	;	73	33	27	1B	11 011
28	\	134	<	74	34	28	1C	11 100
29	]	135	=	75	35	29	1D	11 101
30	^	136	>	76	36	30	1E	11 110

Note: Instruments usually have address switches in binary or hexadecimal.

MR-S-1637-81

Connect the bus cables from one instrument to the next, using either linear or star configurations, as shown in Figure 3–6. The connectors on the bus cables can be fastened securely to each other with the captive locking screws that are part of the connector assembly.

Figure 3–6: Linear and Star Cable Connections



Although any number of instruments can be connected in a star configuration, the combined weight of many connectors could place excessive stress upon the instrument connector supporting them. It is advisable to avoid stacking connectors more than three or four high.

3.3.2 Maximum System Cable Length

If a system is to function properly, certain voltage levels and timing relationships must be maintained on the IEEE bus. Since these are affected by cable length and number of instruments connected to the bus, the following constraints must be observed when installing a system. The total cable length for the system must be no greater than 2 meters (6.6 feet) times the number of instruments (including the IBV11–A) connected to the bus, up to a maximum of 20 meters (66 feet). For example, if there are 3 instruments in your system, the maximum cable length is (3 + 1) * 2 = 8 meters (26.4 feet). If there are 12 instruments in your system, the maximum cable length is 20 meters (66 feet), since (12 + 1) * 2 = 26 meters, which is greater than 20 meters.

3.3.3 Instruments in Your System

If the total number of instruments in your system is N, the number of instruments powered on must be [(N/2) + 1]. Furthermore, each IBV11–A/IB11 in your system must be counted as part of the total number of instruments, N.

Chapter 4
System Software Installation and Operation Procedures

This chapter explains how to use the Instrument Bus Subroutines with your FORTRAN programs under the RT–11 operating system. The chapter outlines the procedures to install, test, and use the Instrument Bus Subroutines software. The chapter also describes the general use of RT–11 commands and utilities needed to install the software.

Because this chapter is intended only as a guide to program development, it does not include detailed information about the FORTRAN programming language, the FORTRAN compiler, or the RT–11 operating system. Therefore, read the chapter to familiarize yourself with the steps you must follow to use the Instrument Bus Subroutines with your own programs. Then see the RT–11 or FORTRAN IV documentation referenced in each section for more information on those topics.

4.1 Preparing to Use the Instrument Bus Subroutines

The package of Instrument Bus Subroutines contains a library of subroutines called IBLIB.OBJ, the IBS verification program called IBSVER.FOR, and four prebuilt IB device drivers called IBMNC.SYS, IBNMNC.SYS, IBXMNC.SYS, and IBXNMC.SYS. You should choose only one of the IB device drivers as follows:

- Use IBMNC.SYS if you have a standard MINC or DECLAB–11/MNC system and if you are using either the SJ or the FB distributed system monitors.
- Use IBNMNC.SYS if you have any other standard PDP–11 system and if you are using either the SJ or the FB distributed system monitors.

- Use IBXMNC.SYS if you have a standard MINC or DECLAB–11/MNC system and if you are using an XM (extended-memory) system monitor, generated for device timeout support and no error logging support.
- Use IBXNMC.SYS if you have any other standard PDP–11 system and if you are using an XM (extended-memory) system monitor, generated for device timeout support and no error logging support.

Before you can use the Instrument Bus Subroutines in your FORTRAN programs, you must install them on your system and then test them to verify that they are properly installed and that they work correctly. Sections 4.2 through 4.4 explain how to install and test your Instrument Bus Subroutines software and how to use them with your FORTRAN programs.

4.2 Installation Requirements

Program development instructions in this chapter require that:

1. All files acted upon by system programs are on the default device, DK, unless a device is specified in a command.
2. You are using the default file types which are:

.FOR	FORTRAN source files
.OBJ	object files and object library files
.SAV	image (executable) files

3. The FORTRAN compiler, FORTRA.SAV, has already been built and resides on the system device.
4. The FORTRAN Object Time System (OTS) has been built and added to the system library, SYSLIB.OBJ, which resides on the system device, SY.
5. The FORTRAN compiler and Object Time System have been successfully tested and verified using the DIGITAL–supplied DEMO.FOR FORTRAN installation/verification test program.
6. All system programs are on the system device, SY.
7. All necessary operating systems and device drivers are installed and reside on the system device, SY.

4.3 Installing your Instrument Bus Subroutines Software

The information that follows explains how to install your Instrument Bus Subroutines software.

4.3.1 Copying the Distribution Kit

The first step in the installation procedure is to copy your Instrument Bus Subroutines software distribution kit. Always keep several copies of any software that cannot be easily re-created. All storage media can be adversely affected by environmental conditions, vandalism, and human error. Therefore, it is good practice to copy (back up) your distribution kit. After you have done so, store the kit in a safe place and use it only when you need to make copies of your software. Use a copy for the following procedure.

The distribution kit containing the Instrument Bus Subroutines is on a single volume with a file structure that RT–11 can read. To copy the distribution kit, do the following:

1. Load the distribution volume.
2. Load an initialized, blank storage volume. (See the *RT–11 System User's Guide* for information about using the INITIALIZE command to initialize your blank storage volume.)

   NOTE

   If you are using RK05 disks you must format each disk before you initialize it. If you are using RX02 drives, you must format each floppy disk to be either single density or double density before you initialize it. See the *RT–11 System User's Guide* for information about using the utility program FORMAT to format your disks or floppies.

3. Use the SQUEEZE command with the /OUTPUT switch to copy files from your distribution volume to your blank storage volume. Using the SQUEEZE command with the /OUTPUT switch transfers all the files from the input volume (your distribution volume) to the beginning of your output volume (your storage volume) and consolidates all the empty space on the output volume at the end of that volume. Use the SQUEEZE command with the /OUTPUT switch to copy your distribution volume rather than the COPY command. The COPY command does not copy the protection code of files.

After the monitor prompt appears type:

```
SQUEEZE/OUTPUT:dvn:  dvm:(RET)
```

where:

/OUTPUT	is a switch that copies all files from the input device to the output device
dv:	is a physical device name
m	is the unit number of the device containing the distribution volume
n	is the unit number of the device containing your blank storage volume

For more information about system commands and copying files, see the *RT–11 System User's Guide.*

4.3.2 Copying with Only Two Mass Storage Devices

If your system has only two mass storage devices, you cannot copy the distribution volume directly to another volume. This problem occurs because the volume containing the RT–11 operating system occupies one of your storage devices.

Therefore, to copy your distribution volume under these circumstances, use both the /OUTPUT and the /WAIT switches with the SQUEEZE command. The /WAIT switch causes a pause before copying begins so that you can remove the system volume from the system device. Then you can load the distribution volume in the system device and begin copying files.

See the *RT–11 System User's Guide* for information about how to use the /OUTPUT and /WAIT switches with the SQUEEZE command to copy your distribution volume if you have only two mass storage devices.

4.3.3 File Protection

All the files in the distribution kit have been protected. If you want to delete or supersede a protected file, you must first change its protection code. To do this, rename the file using the /NOPROTECTION switch with the RENAME command. To protect an unprotected file, use the /PROTECTION switch with the RENAME command.

Never remove the protection of files in your distribution kit. Remove protection only on copies of your software.

4.3.4 Making Corrections

Now make any corrections to your software that may be necessary. You can find these corrections in the RT–11 Software Dispatch along with instructions on how to make the corrections. Since this version of IB represents a newly-released version, you only need to make corrections published in the dispatch for this version of IB.

Never make corrections to your distribution kit. Use only copies of your distribution kit to make corrections. After you make any corrections, you should copy your software again.

4.3.5 Determining if You Need to Build a New IBV11–A Device Handler and IBS Subroutine Library

If you have rebuilt your RT–11 monitor using the RT–11 system generation procedure, or if you do not have a standard IBV11–A device configuration, you must build a new device handler and a new IBS Subroutine Library before you can proceed.

There are two standard IBV11–A device configurations for the Instrument Bus Subroutines software. If you have a MINC or a DECLAB–11/MNC system, the standard IBV11–A device configuration is one in which:

1. There is only one IBV11–A unit.
2. The vector address of that unit is set at location 420(octal).
3. The Control Status Register (CSR) address of that unit is set at 171420(octal).

If you have any other PDP–11 system, the standard IBV11–A device configuration is one in which:

1. There is only one IBV11–A unit.
2. The vector address of that unit is set at location 420(octal).
3. The Control Status Register (CSR) address of that unit is set at 160150(octal).

You cannot use any of the IBV11–A device handlers or the IBS library distributed in your Instrument Bus Subroutines software kit if your system fails to conform to either standard device configuration defined above. Thus, you must build a new IBV11–A device handler and a new IBS subroutine library if at least one of the following is true:

1. You plan to use the device handler with an RT–11 SJ, FB, or XM monitor created during the RT–11 system generation process.
2. Your system has multiple IBV11–A units.
3. Your system has a single IBV11–A unit but its vector address is not set to 420, or its CSR address is not set to 171420 (for MINC and DECLAB–11/MNC users) or to 160150 (for other PDP–11 users).

   If you need to build a new device handler and subroutine library, see Appendix C before going any further, and follow the procedure given there to build a new device handler and subroutine library.

4.3.6 Copying, Installing and Loading your IB Device Handler

Before you can use the IB handler, it must reside on your system volume. Therefore, you need to copy one of the distributed handlers (or your newly created handler) and the distributed library (or your newly created library) to your system volume. Then you need to install and load the handler on your system.

If you already have an IB handler on your system volume, remove it before you perform the copy procedure. After the monitor prompt is displayed type:

```
UNLOAD IB(RET)
REMOVE IB(RET)
```

Now copy the appropriate IB handler (device driver) to your system volume. If you can use one of the distributed handlers, copy the one that corresponds to your particular hardware/software system. (See Section 4.1 to determine which of the distributed handlers corresponds to your particular hardware/software system.) Whichever distributed handler you copy, give it the name IB.SYS if you are running under either the SJ or FB system monitors, or give it the name IBX.SYS if you are running under the XM (extended-memory) system monitor. After the monitor prompt is displayed, type one of the following:

```
COPY/SYSTEM dvn:IBMNC.SYS dvm:IB.SYS(RET)
```

or

```
COPY/SYSTEM dvn:IBNMNC.SYS dvm:IB.SYS(RET)
```

or

```
COPY/SYSTEM dvn:IBXMNC.SYS dvm:IBX.SYS(RET)
```

or

```
COPY/SYSTEM dvn:IBXNMC.SYS dvm:IBX.SYS(RET)
```

where:

/SYSTEM	is the switch that allows you to copy system files
m	is the unit number of the system device
n	is the unit number of the device containing IBMNC.SYS or IBNMNC.SYS or IBXMNC.SYS or IBXNMC.SYS

If you built a new device handler, the procedure you followed in Appendix C creates a file called IB.SYG which must be copied to your system volume with the proper name. After the monitor prompt appears, type:

```
COPY/SYSTEM dvn:IB.SYG dvm:driv.SYS(RET)
```

where:

/SYSTEM	is the switch that allows you to copy system files
m	is the unit number of the system device
n	is the unit number of the device containing your newly created IB handler, IB.SYG
driv	is IB for SJ or FB operating systems and IBX for the XM operating system

When you copy the appropriate IB handler to your system volume, the RT–11 utility program PIP prints the following warning message on your terminal:

```
?PIP-W-Reboot
```

You can ignore this message as long as you unload and remove any IB handler present on your system before you performed the copy procedure. Now install your IB handler. The following format is the same for SJ, FB, and XM system monitors. After the monitor prompt appears type:

```
INSTALL IB(RET)
```

Each time you boot your system, the IB handler will probably be installed automatically. RT–11 automatically installs each handler on its system each time you boot if the following conditions are true:

1. the handler resides on the system volume
2. the device for the handler is physically connected to the hardware system
3. the monitor's device table has enough slots for each type of device physically connected to the hardware system.

Distributed RT–11 monitors contain 16 device slots. If your hardware system has fewer than 16 different types of devices physically connected to it when it is booted, the IB handler will be installed automatically as long as the IBV11–A unit itself is physically connected to the hardware system. If all of your device slots are occupied, you can remove a device handler to make a device slot available for the IB handler. If you cannot remove any of your device handlers, and if you do not have any extra device slots, you will have to regenerate your RT–11 monitor. See Section 3.8.13 of the *RT–11 Installation and System Generation Guide* for more information.

Although RT–11 automatically installs the IB handler when you boot the system, you must still load the handler before you can use it. To load the IB handler, use the following command. The following format is the same for the SJ, FB, and XM system monitors. After the monitor prompt appears type:

```
LOAD IB(RET)
```

You must load the IB handler each time you boot the system or you cannot run any program using the Instrument Bus Subroutines. If you are going to use the IB handler often, you can avoid loading it each time you boot your system by including the LOAD command in the appropriate start-up command file for your monitor. Start-up command files are called STARTS.COM for the SJ monitor, STARTF.COM for the F/B monitor, and STARTX.COM for the XM monitor. Remember that you cannot load the IB handler before it has been installed. Therefore, you should include the LOAD command in a start-up command file only if you are sure that your handler will be installed automatically each time you boot your system.

For more information about copying files, start-up command files and installing and loading devices, see the *RT–11 System User's Guide* and the *RT–11 Installation and System Generation Guide.*

4.4 Testing your Instrument Bus Subroutines Software

After you install your Instrument Bus Subroutines software (see Section 4.5.1), test your software to verify that you performed the installation procedure correctly and that your IB software was delivered in good working order.

To test your IB software, run the test program IBSVER.FOR which is part of your Instrument Bus Subroutines distribution kit. To run IBSVER.FOR, do the following:

1. Make sure that the installation requirements listed in Section 4.2 are true for your system
2. Make sure that the following assumptions are also true:
   - The test program IBSVER.FOR and the library of Instrument Bus Subroutines, IBLIB.OBJ are on the default device.
   - The default device has approximately 60 blocks on it for the files IBSVER.OBJ and IBSVER.SAV that you will create in the process of running the test program.
   - For the purposes of the test program the device handler, IB.SYS or IBX.SYS has not yet been loaded.
   - No devices are connected to IBV11–A/IB11 instrument bus.
3. Run IBSVER.FOR by typing all the commands that appear in red in the following program sample. Type each command exactly as it appears and wait for the computer's response before you type the next command. The computer's response appears in black.

```
.LOAD IB[RET]

.FORTRAN IBSVER[RET]
.MAIN.
SERVE

.LINK IBSVER,IBLIB[RET]

.RUN IBSVER[RET]
```

IBSVER checks IB software functions by intentionally creating non-fatal error conditions. If the software responds to these conditions with appropriate error messages, the chances are good that it is operating correctly. Compare the messages the program displays on your terminal with the paradigm below. The results should match. If they do not match, and if you have not made any typing errors, you may not have received a reliable copy of your Instrument Bus Subroutines distribution kit. Contact DIGITAL for further information.

```
                    IBS-11 V2.1 Verification Program

This procedure assumes that IB.SYS (or IBX.SYS) has been INSTALLED
and LOADED in this system. Also, this procedure assumes that no de-
vices are connected to your IBV11-A/IB11 instrument bus.

The first part of this test calls routines that do not depend on any
device and thus should produce no error messages.

Error in routine IBTIMO: No timeout support available

IBREMO should return minus one at this point. IBREMO =   -1

At this point non-fatal error messages should begin to appear.

                    ****EXPECTED ERROR MESSAGES****
                    -------------------------------

Error in routine IBTIMO: No timeout support available

Error in routine IBLNR: No default listener list available

Error in routine IBSEND: No valid listener on the bus

Error in routine IBSEOI: No valid listener on the bus

Error in routine IBRECV: No valid listener on the bus

Error in routine IBXFER: No valid listener on the bus

Error in routine IBASND: No valid listener on the bus

Error in routine IBARCV: No valid listener on the bus

Error in routine IBAXFR: No valid listener on the bus

Error in routine IBFREE: No valid listener on the bus

Error in routine IBGTL: No valid listener on the bus

Error in routine IBSDC: No valid listener on the bus

Error in routine IBGET: No valid listener on the bus

Error in routine IBCMD: No default listener list available

Error in routine IBSTS: No valid listener on the bus

Error in routine IBSPL: No valid listener on the bus

Error in routine IBPPE: No default listener list available

Error in routine IBPPD: No default listener list available

Error in routine IBPPU: No valid listener on the bus

Error in routine IBUNL: No valid listener on the bus

Error in routine IBUNT: No valid listener on the bus

Error in routine IBLLO: No valid listener on the bus

Error in routine IBDCL: No valid listener on the bus

Error in routine IBUNIT: Invalid IBV11-A unit number


IBS-11 VERIFICATION PROCEDURE SUCCESSFUL!

STOP -- END IBS VERIFY
```

4.5 Creating a Program that Calls the Instrument Bus Subroutines

After you install your Instrument Bus Subroutines software and test it to verify that it works correctly, you are ready to use the subroutines in your own FORTRAN programs.

To create a FORTRAN program that calls the Instrument Bus Subroutines, do the following:

1. Write and check your program.
2. Use one of the RT–11 editors, such as EDIT, to enter your program into a source file. After the monitor prompt appears type:

   ```
   EDIT/CREATE prog.FOR(RET)
   ```

   where:

/CREATE	is the switch that allows the editor to create a new file
prog	is the name of your FORTRAN source program
.FOR	is the file type for a FORTRAN source file

   For information about entering the text of your file, making changes, and typing or printing the file, see the *Introduction to RT–11* and the *RT–11 System User's Guide.*

   NOTE

   Always specify a file type when you use an RT–11 editor. RT–11 editors do not use default file types. In this case, give your source file the type .FOR since that is the default file type the FORTRAN IV compiler uses when it compiles your program.

3. Use the FORTRAN IV compiler to create an object file of your program. After the monitor prompt is displayed, type:

   ```
   FORTRAN prog(RET)
   ```

   where:

prog	is the name of your FORTRAN source program

4. Use the RT–11 linker to link the object file of your program with IBLIB.OBJ, the Instrument Bus Subroutines library. After the monitor prompt is displayed, type:

   ```
   LINK prog,IBLIB(RET)
   ```

where:

prog	is the name of your FORTRAN program
IBLIB	is the name of the Instrument Bus Subroutines library

If you want to receive short error messages (see Section 9.1), type the commands in red in the following example:

```
.LINK/INCLUDE prog,IBLIB(RET)
Library Search? IB$SRT(RET)
Library Search?(RET)
```

where:

/INCLUDE	is the switch that you use to include a global symbol from a library
IB$SRT	is the name of the global symbol that you want to include from your library

5. Run the program. Remember that IB.SYS or IBX.SYS must be installed and loaded before you can run the program. After the monitor prompt is displayed, type:

```
RUN prog(RET)
```

where:

prog	is the name of your executable program

For more information about compiling and linking FORTRAN IV programs, see the *RT–11 System User's Guide* and the *RT–11/RSTS/E FORTRAN IV User's Guide.*

4.5.1 Using Libraries

You receive the IB subroutines as a pre-built object library called IBLIB.OBJ. The library makes it easier to organize and link the subroutines. Each time you call an IB subroutine, it may call several general-purpose subroutines as well. If you use the IB subroutines as a library, you do not need to remember the relationships between them and the general-purpose subroutines. Each time you link a program requiring any IB subroutine, you only need to include the name of the library in the link command to call all the necessary subroutines.

If you plan to use the IB subroutines often, you can make them easier to use by placing IBLIB.OBJ in the RT–11 system library, SYSLIB.OBJ. If you place IBLIB in SYSLIB, you do not need to specify the name of the library in the link command at all. The RT–11 linker always searches the system library for a subroutine or function needed by a program even though it was not named in the link command.

You can add IBLIB to SYSLIB using the RT–11 librarian program, LIBR.SAV. However, there are considerations you should be aware of when you attempt to do this. Adding IBLIB (or any other file) to SYSLIB causes the entire SYSLIB file to be rebuilt. In particular, this means that any global symbols removed from SYSLIB's directory during the last library build procedure must be removed again when you add IBLIB (or any other file) to SYSLIB.

Global symbols are removed from SYSLIB's directory on at least two occasions:

1. When SYSLIB.OBJ is built for distribution. At this time the symbol $OVRH is removed from the library directory.
2. When you add the FORTRAN IV OTS library to the system library. This occurs if you answered "Y" to the first question asked by the OTS generation program, OTSGEN.SAV, distributed with your FORTRAN IV/RT–11 V2.5 software. At this time, two additional symbols are removed: $ERRS and $ERRTB.

Also, you must remove any global symbols previously removed from IBLIB's directory (or from the directory of any other library you add to an existing library). In this case, remove the global symbol IB$ERR.

To add IBLIB to a system library containing the distributed SYSLIB.OBJ file and the FORTRAN IV OTS library, use the LIBRARY command with the /REMOVE switch. The /REMOVE switch allows you to remove a global symbol from a library directory. After the monitor prompt is displayed, type the commands in red:

```
LIBRARY/REMOVE dvn:SYSLIB dvm:IBLIB.OBJ(RET)
Global? $OVRH(RET)
Global? $ERRS(RET)
Global? $ERRTB(RET)
Global? IB$ERR(RET)
Global?(RET)
```

where:

/REMOVE	is the switch that you use to remove a global symbol from a library.
m	is the number of the device on which IBLIB is stored.
n	is the number of the device on which SYSLIB is stored.

NOTE

If you do not remember the names of global symbols to remove when you execute this command, a number of warning messages appear equal to the number of globals you failed to remove from SYSLIB's or IBLIB's directory. Each message specifies one of the global symbols you failed to remove. An example of the warning message is:

```
?LIBR-W-Illegal insert of $OVRH
```

When this happens, each global symbol specified in an error message is now defined in SYSLIB's directory more than once. You may get unexpected and undesirable results when you link programs to libraries with directories containing global symbols that are defined more than once.

For more information about libraries, see the *RT–11 System User's Guide.*

Chapter 5
Introductory Programming Techniques

Data or programmed instructions can be sent as ASCII characters from one instrument on the IEEE bus to other instruments on the bus. This data is often coded as ASCII characters.

NOTE

> Because ASCII characters are the usual data format used, this manual refers to bytes of data as "characters".

Since the IBV11–A/IB11 is an instrument, it can send and receive such ASCII characters. An instrument sending data or programmed instruction is called the "talker". The instrument or instruments receiving data or programmed instructions from the talker are called "listeners." At any given time only one instrument can talk, but any number of instruments can listen.

The IB routines described in this chapter are used to send and receive data between instruments on the IEEE bus. They are used in FORTRAN programs to direct the operation of your instrument system. You can accomplish many tasks on the instrument bus with these routines. They may be the only routines you need. These routines are an important part of any program you write, so become thoroughly familiar with them before you begin to learn about other IB routines in the chapters that follow.

5.1 Routine Calling Format

Any IB routine can be called by a FORTRAN CALL statement. Each routine description in this document includes the format of that routine in a FORTRAN CALL statement. For example, the format of the routine IBRECV, described in this chapter, is shown in the following example as a guide to the way it would appear in a CALL statement.

```
CALL IBRECV (array,len-limit[,[tkr][,lnrs]])
```

The routine format consists of the routine name, IBRECV, and a sequence of argument names within parentheses, separated from one another by commas.

The argument names are in lower case letters to indicate that appropriate values (variables, numbers, or quoted character strings) should be entered in place of the argument names. When you enter values, you must type the parentheses that enclose them and the commas that separate them. The commas mark the boundaries that define an argument's position in the argument sequence.

Arguments within square brackets are optional. They need not be specified. Do not type the square brackets when entering a value in an optional argument. Type the commas as you would for any argument. The commas that mark the position of each argument in the argument sequence need not be typed after the last specified argument. When no value is entered in an optional argument, the IB routine uses a predetermined value called a default value. For example, in the IBRECV CALL statement above, all the following argument specification formats are legal:

```
CALL IBRECV (array,len-limit)
CALL IBRECV (array,len-limit,tkr)
CALL IBRECV (array,len-limit,,lnrs)
CALL IBRECV (array,len-limit,tkr,lnrs)
```

Some IB routines can be used as functions, as well as being called by a CALL statement. When this is the case, the IB routine is assigned an integer value by the IB software. IBRECV can be used as a function. The value assigned to it can then be used in your program by any FORTRAN statement. In the following example, an ellipsis (...), for the sake of brevity, is used in place of the full argument description.

```
TYPE 10, (M(I), I=1, IBRECV(...))
L=1+IBRECV(...)
```

In the second line, IBRECV has been assigned a numeric value, which is added to the integer 1.

When an IB routine can be used as a function, the routine description includes a full function description, format, and the values that are assigned.

5.2 Talker and Listener Addressing

You can designate which instrument is to be talker and which instruments are to be listeners by specifying their addresses in the appropriate arguments of an IB routine. Not all routines require both a talker and a listener. Some routines accept no address specification. See the routine descriptions that follow in this and succeeding chapters.

Only one instrument can talk at a time. Whenever you address an instrument as talker, the previous talker is automatically unaddressed. The new talker is the only instrument that can transmit data. The previous address is removed from the IEEE bus and the instrument must be readdressed as talker before it can talk again. The IB routines remember internally the instrument most recently specified as talker on the default talker list (see Section 5.2.4). In some IB routines, the IBV11–A/IB11 is the talker. In these cases, the IB routines automatically specify the IBV11–A/IB11 as talker, because the IBV11–A/IB11 does not have an instrument address. The default talker list remains unchanged.

Usually when you address one or more instruments as listeners in an IB routine, all previous listeners are automatically unaddressed. Their addresses are removed from the default listener list (see Section 5.2.4), and the new listeners' addresses are entered. Only the new listeners can receive data. You must readdress the instruments previously addressed as listeners before they can listen again. The IB routines remember internally the instruments most recently specified as listener on the default listener list. In some routines the IBV11–A/IB11 is the listener. In these cases the IB routines automatically specify the IBV11–A/IB11 as listener, because the IBV11–A/IB11 does not have an instrument address.

5.2.1 Addressing with Primary and Secondary Addresses

Each instrument that can be used with the IEEE bus has a number in the range 0 through 30 which identifies it uniquely. This number is known as its primary address, and corresponds to the setting of the address switches on that instrument. (Many instrument addresses are selectable.) The address for each instrument must always be used to designate that instrument as talker or listener. Note, however, that IB only supports a primary address of zero in its ASCII translation (see Section 5.2.2.).

Some instruments have more than one function. These functions are sometimes designated by secondary addresses, which fall in the range 0 through 30. Secondary addresses are specified in address arguments as the corresponding decimal values 200 through 230. They must always be preceded by the primary address of the instrument with which they are associated.

When you want to designate an instrument as talker, enter its primary address in the argument that specifies the talker. If you want to designate one of its alternate functions as talker, enter the primary address of that instrument, followed by a decimal value (200 through 230) corresponding to its secondary address.

Similarly, when you want to designate an instrument as listener, enter its primary address in the argument that specifies listeners. If you want to designate one of its alternate functions as listener, enter the primary address of that instrument followed by a decimal value (200 through 230) corresponding to the secondary address.

Examples:

Assume that a system contains one instrument with primary address 5 and another with primary address 6, and that instrument 5 has an alternate function that can be enabled by the secondary address 13. Both instruments can be talkers or listeners.

The following examples specify instruments as talker or listener with primary and secondary addresses, using the conventions described in Section 5.2. The address arguments are last in the argument sequence. Since we are interested here only in addressing, no other elements of the call statement are shown, but are indicated by three dots (ellipsis) to the left of the first address argument.

In each example, the argument name is given first, as it would appear in the routine format (see Section 5.1); then the specific values are substituted for the argument names.

1. Specify instrument 5 as talker. Use the primary address.

(..., tkr)	[the argument name]
(..., 5)	[the specific value 5 has been substituted for the argument name tkr.]

2. Specify instrument 6 as talker. Use the primary address.

   (..., tkr)
   (..., 6)

3. Specify the alternate function of instrument 5 as talker. Use the primary address, followed by the secondary address, separating them with a comma. Specify the secondary address 13 by using its corresponding value 213.

   (..., tkr)
   (..., 5, 213)

4. Specify instrument 6 as listener. Use the primary address.

   (..., lnrs)
   (..., 6)

5. Specify instruments 6 and 5 as listeners. Use the primary addresses.

   (..., lnrs)
   (..., 6, 5)

6. Specify instrument 5 with its alternate function 13 and instrument 6 as listeners,

   (..., lnrs)
   (..., 5, 213, 6)

7. Specify instrument 6 as talker and instrument 5, with its alternate function, as listener. Only the first primary address is recognized as specifying a talker.

   (..., tkr, lnrs)
   (..., 6, 5, 213)

8. Specify instrument 5 with its alternate function as talker and instrument 6 as listener.

   (..., tkr, lnrs)
   (..., 5, 213, 6)

The values that you entered in example 8 look exactly like those entered in example 6 above, when you specified instrument 5, with its alternate function, and instrument 6 both as listeners. Each IB routine interprets the arguments according to its own format. In example 6, the routine format calls for only a listener specification. Both addresses 5, 213 and 6 are entered in the lnrs argument. In example 8, the routine format calls for both a talker and a listener specification. The address 5, 213 is entered in the tkr argument, and the address 6 is entered in the lnrs argument.

5.2.2 Addressing with ASCII Characters

You can also address instruments by using ASCII characters. When you designate an instrument as talker, the IB routines automatically translate its primary address into the ASCII character that directs that instrument to talk. This character is in the range @ through ^, which corresponds to the decimal range 64 through 94, or octal 100 through 136. The IBV11–A/IB11 accepts these ASCII characters as valid talker addresses that correspond to the primary addresses 0 through 30. ASCII characters used as addresses must appear between single quotation marks (for example, 'B').

When you designate an instrument as listener, the IB routines automatically translate its primary address into an ASCII character that directs that instrument to listen. This character is in the range SP through >, which corresponds to the decimal ranges 32 through 62 or octal 40 through 76. The IBV11–A/IB11 accepts these ASCII characters as valid listener addresses that correspond to primary addresses 0 through 30.

NOTE

> IB supports a primary address of zero only in its ASCII (or the equivalent decimal or octal) translation. The talker specification for a primary address of zero is "@" (the at sign — decimal 64, octal 100), and the corresponding listener specification is "SP" (space — decimal 32, octal 40).

The IB routines also translate secondary address specifications into ASCII characters in the range apostrophe (') through tilde (~). These characters correspond to the decimal ranges 96 through 126, or octal 140 through 176. The IBV11–A accepts these ASCII characters as valid secondary addresses that correspond to decimal values 200 through 230.

Examples:

The following examples illustrate the use of ASCII characters for talker and listener addressing. The characters correspond to the primary and secondary addresses used above.

	ASCII Characters	
Primary and Secondary Addresses	Talker	Listener
5,213	E,m	%,m
6	F	&

1. Specify instrument 5 as talker, using its corresponding ASCII character.

   (..., tkr)
   (..., 'E')

2. Specify instrument 6 as talker, using its corresponding ASCII character.

   (..., tkr)
   (..., 'F')

3. Specify the alternate function of instrument 5 as talker, using its corresponding ASCII characters.

   (..., tkr)
   (..., 'Em') or (..., 'E', 'm')

4. Specify instrument 6 as listener, using its corresponding ASCII character.

   (..., lnrs)
   (..., '&')

5. Specify instruments 6 and 5 both as listeners, using their corresponding ASCII characters.

   (..., lnrs)
   (..., '&%') or (..., '%&') or (..., '&', '%')

6. Specify instrument 5 with its alternate function, and instrument 6 as listeners, using their corresponding ASCII characters.

   (..., lnrs)
   (..., '%m&') or (..., '%m', '&') or (..., '%', 'm', '&')

7. Specify instrument 6 as talker and instrument 5 with its alternate function as listener, using their corresponding ASCII characters.

   (..., tkr, lnrs)
   (..., 'F', '%m') or (..., 'F%m') or ('F', '%', 'm')

8. Specify instrument 5 with its alternate function as talker and instrument 6 as listener, using their corresponding ASCII characters.

   (..., tkr, lnrs)
   (..., 'Em', '&') or (..., 'Em&') or (..., 'E', 'm', '&')

In examples 7 and 8, note that different ASCII characters are used to address the same instrument to talk and to listen. If you use ASCII characters, you are doing the translation that the routines do automatically when they change primary and secondary addresses into proper talker or listener addresses. However, if you use an ASCII character which would normally denote a talker to designate a listener, the IB routine translates this specification into the proper listener character. For example, although the ASCII character E is actually the ASCII code telling instrument 5 to talk, you can use E to denote instrument 5 as a listener. In this case, the IB routine translates the E into %. Similarly % may also be used to specify instrument 5 as either a listener or a talker.

5.2.3 Addressing with Byte Arrays

You can address instruments using byte arrays. Each element of an array contains an address. A list of addresses must always end with a zero byte (0).

Examples:

These examples use byte arrays for talker and listener addressing. The values in the byte array may be integers or ASCII characters corresponding to primary or secondary addresses.

1. Specify instrument 5 as talker, using a byte array.

```
(..., tkr)

BYTE ADDR(2)                  or   BYTE ADDR(2)
DATA ADDR /5, 0/                   DATA ADDR /'E', 0/
.                                  .
.                                  .
.                                  .
(...,ADDR)                         (...,ADDR)
```

2. Specify instrument 6 as talker, using a byte array.

```
(..., tkr)

BYTE ADDR(2)
DATA ADDR /6, 0/
.
.
.
(..., ADDR(1))
```

3. Specify the alternate function of instrument 5 as talker using a byte array.

   (..., tkr)

```
BYTE ADDR(3)                 or   BYTE ADDR(3)
DATA ADDR /5, 213, 0/             ADDR(1)=5
.                                 ADDR(2)=213
.                                 ADDR(3)=0
.                                 .
(..., ADDR)                       .
                                  .
                                  (..., ADDR)
```

4. Specify instrument 6 as listener, using a byte array.

   (..., lnrs)

```
BYTE ADDR(2)                 or   BYTE ADDR(2)
DATA ADDR /'&', 0/                ADDR(1)='&'
                                  ADDR(2)=0
.                                 .
.                                 .
.                                 .
(..., ADDR)                       (..., ADDR)
```

5. Specify instruments 6 and 5 both as listeners, using a byte array.

   (..., lnrs)

```
BYTE ADDR(3)                 or   BYTE ADDR(4)
DATA ADDR /6, '%', 0/             DATA ADDR /6, 0, 5, 0/
.                                 .
.                                 .
.                                 .
(..., ADDR)                       (..., ADDR(1), ADDR(3))
```

6. Specify instrument 5 with its alternate function and instrument 6 as listeners, using a byte array.

   (..., lnrs)

```
BYTE ADDR(4)                 or   BYTE ADDR1(3) ADDR2(2)
DATA ADDR /5, 213, 6, 0/          DATA ADDR1/5,213,0/
                                  DATA ADDR2 /6, 0/
.                                 .
.                                 .
.                                 .
(..., ADDR)                       (..., ADDR1, ADDR2)
```

7. Specify instrument 6 as talker and instrument 5 with its alternate function as listener, using a byte array.

   ```
   (..., tkr, lnrs)

   BYTE ADDR(4)                    or   BYTE TADDR(2), LADDR(3)
   DATA ADDR /'F', '%', 'm', 0/         DATA TADDR /'F', 0/
                                        DATA LADDR /'%', 'm', 0/
   .                                    .
   .                                    .
   .                                    .
   (..., ADDR)                          (..., TADDR, LADDR)
   ```

8. Specify instrument 5 with its alternate function as talker and instrument 6 as listener, using a byte array.

   ```
   (..., tkr, lnrs)

   BYTE ADDR(4)                    or   BYTE ADDR(5)
   DATA ADDR /5, 213, 6, 0/             DATA ADDR /5, 213, 0, 6, 0/
   .                                    .
   .                                    .
   .                                    .
   (..., ADDR)                          (..., ADDR, ADDR(4))
   ```

5.2.4 Default Talker and Listener Lists

Default talker and listener lists are used whenever a talker or listener specification is required but has not been specified, or whenever transmission is reestablished after an interrupt.

Instruments last specified as listeners in an IB routine are remembered internally as listeners on the default listener list. Commands are sent on the IEEE bus addressing those instruments to listen.

When you specify instruments as listeners in a later IB routine, they replace the previous listeners on the default listener list and on the IEEE bus. If a listener is required in an IB routine but none is specified, the instruments on the default listener list (those last specified as listeners) are addressed to listen if their addresses are no longer on the IEEE bus.

The instrument last specified as talker in an IB routine is remembered internally by the IB routines as talker on the default talker list. A command is sent on the IEEE bus addressing that instrument to talk.

When you specify an instrument as talker in a later IB routine, it replaces the previous talker on the default talker list and on the IEEE bus. If a talker is required in an IB routine but none is specified, the instrument on the default talker list (the last one specified as talker) is addressed to talk if its address is no longer on the IEEE bus.

You can add new listeners to the current default listener list by entering minus one (–1) as the first element of the listener specification. This prevents current listeners from being removed from the list. A –1 cannot be entered in any address specification, except as the first element of the listener specification. It can never be used in a talker specification.

When the main program is interrupted by a service request (see Section 8.1.2), default talker and listener lists are established locally within the SRQ routine. The default lists of the main program are neither used nor altered. An error occurs if talker or listener specifications are not given in a routine in which they are required before their respective default lists have been established, either in the main program or locally within the SRQ routine.

Examples:

1. IBGET (see Section 6.3.3) requires a listener specification. The following program segment illustrates use of the default listener list.

   FORTRAN Program   These instruments listen

```
.
.
.
CALL IBGET(1)                    1
CALL IBGET(2)                    2
CALL IBGET( )                    2
CALL IBGET(-1, 3, 212, 6)        2 3,212 6
CALL IBGET                       2 3,212 6
CALL IBGET(4,5)                  4 5
.
.
.
```

2. IBSTS (see Section 7.1.1) is used as a function and requires a talker specification. The following program segment illustrates the use of the default talker list.

   FORTRAN Program   This instrument talks

```
.
.
.
ISTAT=IBSTS(1)                   1
ISTAT=IBSTS(2)                   2
ISTAT=IBSTS( )                   2
ISTAT=IBSTS(3,212)               3,212
ISTAT=IBSTS( )                   3,212
ISTAT=IBSTS(4)                   4
.
.
.
```

3. IBSEND (see Section 5.3.1) requires a listener specification. The IBV11–A/IB11 is automatically the talker. IBXFER (see Section 5.3.4) requires both a talker and a listener specification. Both these routines also require arguments other than addresses: for clarity, no values are entered in these arguments.

```
FORTRAN Program                               This          These
                                           instrument    instruments
                                             talks         listen

.
.
.
CALL IBSEND(msg-array,msg-len,1)           IBV11-A/IB11  1

CALL IBXFER(len-limit,2,3)                 2             3

CALL IBSEND(msg-array,msg-len)             IBV11-A/IB11  3

CALL IBXFER(len-limit,4,205,-1,6)          4,205         3 6

CALL IBSEND(msg-array,msg-len,-1,8,212,7)
                                           IBV11-A/IB11  3 6 8,212 7

CALL IBXFER(len-limit)                     4,205         3 6 8,212 7

CALL IBSEND(msg-array,msg-len,2)           IBV11-A/IB11  2

CALL IBXFER(len-limit,1)                   1             2

CALL IBXFER(len-limit,,3)                  1             3

CALL IBXFER(len-limit,,-1,6)               1             3 6

.
.
```

5.3 Data Transmission

The IEEE bus provides for data transmission among all instruments connected to the bus. IB routines provide three types of transmission, as follows:

1. They send data or programmed instruction to instruments.
2. They receive data from an instrument, storing the data in an array you specify.
3. They transfer data from one instrument to another with no data storage.

5.3.1 Sending Data (IBSEND)

Use IBSEND to send data from the computer to specified listeners. Use IBSEOI if an EOI is required. The IBV11–A/IB11 is the talker. The previous talker is automatically removed from the IEEE bus, but the talker default list remains unchanged. It is used automatically during the next command that requires a talker, if one is not specified.

Call IBSEND as a subroutine.

CALL IBSEND (msg-array [, [msg-len] [, lnrs]])

where:

msg-array	The message that you want to send in single quotes, or the name of the array in which that message is stored. If format conversion of a numerical value is required, use the FORTRAN statement ENCODE (see the *RT–11/RSTS/E FORTRAN IV User's Guide*) before sending the message.
msg-len	The number of characters you want to send from the array up to a maximum of 65,534 (see Appendix F for specifying numbers greater than 32,767). If msg-len is -1, the contents of the array are sent until a null byte (0) is reached. The null byte terminates the transmission but is not sent. Specify -1 for msg-len when you send a message in quotes or when the last character in the message is followed by a null byte. If msg-len is not specified, the effect is the same as specifying -1. If msg-len is 0, no message is sent. The listeners specified in this routine replace those previously on the default listener list. This can be used to change the default listener list without sending a message. If msg-len is greater than 0, it specifies the number of characters to be sent. The character 0 is sent like any other character. It does not terminate the message.
lnrs	The optional listener specification. If you do not specify this argument, IBSEND uses the default listener list.

Example:

This program accepts character strings from the terminal and sends those strings to instrument 6.

```
        BYTE MESSAG (20)
        INSTR = 6
1       TYPE 1001,INSTR
        ACCEPT 1002, NCHAR, MESSAG
        IF (NCHAR.GT.20) GO TO 10
        CALL IBSEND (MESSAG, NCHAR, INSTR)
        GO TO 1
10      TYPE 1003
        GO TO 1
1001    FORMAT (' Enter characters to be sent to instrument ',
     1  I2,'? ',$)
1002    FORMAT (Q, 20A1)
1003    FORMAT (1X,'Too many characters, try again.')
        END
```

5.3.2 End or Identify (IBSEOI)

IBSEOI is like IBSEND, with one exception: when IBSEOI is used, the EOI (end or identify) bit of the IEEE bus is set when the last byte of data is sent (see the *IB11 UNIBUS to IEEE Instrument Bus Interface Installation Manual*). Many instruments require this so that they can identify the end of the message. See the operation manual for each instrument.

CALL IBSEOI (msg-array [, [msg-len] [, lnrs]])

where:

msg-array	The message that you want to send in single quotes, or the name of the array in which the message is stored. If format conversion of a numerical value is required, use the FORTRAN statement ENCODE (see the *RT–11/RSTS/E FORTRAN IV User's Guide*) before sending the message.
msg-len	The number of characters you want to send from the array. If msg-len is −1, the contents of the array are sent until a null byte (0) is reached. The null byte terminates the transmission but is not sent. Specify −1 for msg-len when you send a message in quotes or when the last character in the message is followed by a null byte. If msg-len is not specified, the effect is the same as specifying −1. If msg-len is 0, no message is sent. The listeners specified in this routine replace those previously on the default listener list. This can be used to change the default listener list without sending a message. If msg-len is greater than 0, it specifies the number of characters to be sent. The character 0 is sent like any other character. It does not terminate the message.
lnrs	The optional listener specification. If you do not specify this argument, IBSEOI uses the default listener list.

Examples:

You want to send the message M23.1,103.5 to instrument 5. This instrument requires that you terminate messages to it with a carriage return (ASCII octal value 15) sent with the EOI bit set. Use example 1 or 2:

1.
```
INSTR=5
CALL IBSEND ('M23.1,103.5', -1, INSTR)
CALL IBSEOI ("15, 1)        !Carriage return is octal 15
```

2.
```
        BYTE MESSAG(5)
        DATA X/23.1/, Y /103.5/
        CALL IBSEND ('M' , , 5)
        ENCODE (4, 1001, MESSAG) X
        CALL IBSEND (MESSAG, 4)
        CALL IBSEND (',')
        ENCODE (5, 1001, MESSAG) Y
  1001  FORMAT (F5.1)
        CALL IBSEND (MESSAG, 5)
        CALL IBSEOI ("15)
```

If a message is broken into more than one IBSEND or IBSEOI call, all calls following the first call should use the default listener list. Use no other IB routines between the calls in which the message begins and ends.

The service request routine waits for any IB calls in progress to complete before it interrupts the main program. If you have specified a service request routine (see Section 8.1.3) do not break a message into more than one IBSEND or IBSEOI call. Doing so could result in an incomplete (and probably useless) message, as the service request routine could interrupt after the message in the first call is transmitted.

Examples 1 and 2 can be used in the absence of a service request routine. Only example 3 should be used when a service request routine is specified.

3.
```
        BYTE MESSAG (13)
        DATA X/23.1/, Y/103.5/
        DATA MESSAG (1)/'M'/ , MESSAG(6)/','/
        DATA MESSAG (12)/"15/, MESSAG(13)/0/
        ENCODE (4, 1001, MESSAG(2)) X
        ENCODE (5, 1001, MESSAG(7)) Y
  1001  FORMAT (F5.1)
        CALL IBSEOI (MESSAG, , 5)
```

5.3.3 Receiving Data (IBRECV)

Use IBRECV to receive data messages through the IBV11–A/IB11 from a specified talker. The IBV11–A/IB11 is always a listener, even when none is specified. The data message is stored in a message array. Use IBRECV as a function.

len = IBRECV (array, len-limit [, [tkr] [, lnrs]])

where:

len — An integer variable that is set equal to the total number of characters actually received by the listeners. This number never exceeds the maximum imposed by len-limit. Data messages will be terminated when:

- The number of data characters reaches the maximum imposed by the len-limit specification.

- The EOI control bit is detected as set when a character is sent by the talker (see the *IBV11–A LSI–11/Instrument Bus Interface User's Manual*).
- A terminating character is received. The terminating character is normally $\boxed{\text{LF}}$ or $\boxed{\text{RET}}$ unless another is defined by IBTERM (see Section 10.2). The terminating character is stored with the rest of the associated message.
- Any error is detected.

array — The name of an array in which data received from the talker is stored. If format conversion to numeric values is required, use the FORTRAN statement DECODE (see the *RT–11/RSTS/E FORTRAN IV User's Guide*) after the data is received. Note that this parameter must be specified; it cannot be defaulted.

len-limit — A positive integer or 0, that specifies the maximum number of characters to be stored in the array. This integer should be no greater than the size of the array. You should specify len-limit to accommodate any anticipated number of characters. A 0 here means that no characters will be received. This argument must be specified. No default value is allowed.

tkr — The optional talker specification. If you do not specify a talker, IBRECV uses the default talker list. If you specify a talker, it replaces the default talker in the default talker list.

lnrs — The optional listener specification. If you do not specify a listener, IBRECV does NOT use the default listener list. Instead, the IBV11–A/IB11 is the ONLY listener. Any default listeners are removed from the IEEE bus. The default listener list remains unchanged. IBRECV and IBARCV are the only routines in which the default listeners list is not used if you do not specify a listener.

To use the current default listener list, specify −1. The default listeners will listen, along with the IBV11–A/IB11.

To add new listeners to the current default listener list, specify −1, and then enter the addresses of the instruments you want to listen. This also applies for any other routine.

To specify listeners, do so without using −1. These new listeners become the default listener list, as in any other routine. These instruments and the IBV11–A/IB11 all listen.

When the full capacity of IBRECV as a function is not required, you can call IBRECV as a subroutine.

CALL IBRECV (array,len-limit[,[tkr][,lnrs]])

The arguments are the same as for IBRECV used as a function.

Examples:

1. Get a maximum of 10 bytes from instrument 5 and type the result on the terminal.

```
        BYTE MESSAG(10)
        .
        .
        .
        N=IBRECV(MESSAG, 10 , 5)
        TYPE 1001, (MESSAGE(I), I=1, N)
1001    FORMAT(1X, 10A1)
        .
        .
        .
```

or

```
        BYTE MESSAG(10)
        .
        .
        .
        TYPE 1001, (MESSAG(I), I=1, IBRECV(MESSAG, 10, 5))
1001    FORMAT(1X, 10A1)
        .
        .
        .
```

2. Suppose that instrument 7 is a printer on which you want to keep a log of all bus data. This program segment allows the printer to listen to the message that instrument 5 is sending to the IBV11–A/IB11.

```
        BYTE MESSAG(10)
        .
        .
        .
        TYPE 1001, (MESSAGE(I), I=1, IBRECV(MESSAG, 10, 5, 7))
1001    FORMAT(1X, 10A1)
        .
        .
        .
```

3. After the multimeter has been sent the string M0T2R0X, it sends a reading every time IBGET is issued (See Section 6.3.3). The data is in the format:

   XXXX+d.ddddE+d$\boxed{\text{RET}}$

   where:

XXXX	information other than the numeric reading
d	a digit 0 through 9
E	the character E
$\boxed{\text{RET}}$	the return character

Assuming that the multimeter is instrument 2, your program to type the average of two readings is:

```
        BYTE READ1(15), READ2(15)
        METER=2
        .
        .
        .
        CALL IBSEND ('MOT2ROX', , METER)
        CALL IBGET (METER)              ! Measure resistance
        CALL IBRECV (READ1, 15, METER)  ! Get reading
        CALL IBGET                      ! Measure again
        CALL IBRECV (READ2, 15)         ! Get reading
        DECODE (10, 1001, READ1(5)) R1
        DECODE (10, 1001, READ2(5)) R2
1001    FORMAT (E10.4)
        AVERAG = (R1+R2)/2
        TYPE 1002, AVERAG
1002    FORMAT (1X, 'Average of the 2 readings is ', E10.4)
        .
        .
        .
```

5.3.4 Transferring Data (IBXFER) on the IEEE Bus

Use the IBXFER function to establish and control the transfer of data among instruments on the IEEE bus. IBXFER does not store this data. Use IBXFER as a function.

len = IBXFER ([len-limit] [, [tkr] [, lnrs]])

where:

len	An integer variable that is set equal to the total number of characters actually received by the listeners. This number will never exceed the maximum imposed by len-limit. Data messages will be terminated when: • The number of data characters reaches the maximum imposed by the len-limit specification. • The EOI control bit is detected as set when a character is sent by the talker (see the *IBV11-A LSI-11/Instrument Bus Interface User's Manual*). • A terminating character is received. The terminating character is normally RET or LF unless another is defined by IBTERM (see Section 10.3). • Any error is detected.
len-limit	A positive integer (or 0) that specifies the maximum number of characters to be transferred. Len-limit should be specified to accommodate any anticipated number of characters. A 0 here means that no characters will be transferred. Default or −1 allows the maximum number of characters (65,535 decimal) to be transferred.

tkr	The optional talker specification. If you do not specify this argument, IBXFER uses the default talker list.
lnrs	The optional listener specification. If you do not specify this argument, IBXFER uses the default listener list.

When the full capabilities of the function are not required, call IBXFER as a subroutine. The integer representing the actual number of data characters received (len) is not available.

```
CALL IBXFER ([len-limit] [, [tkr] [, lnrs]])
```

The arguments are the same as those used for the function IBXFER.

Examples:

If you want up to 10 bytes of data from instrument 5 to go to instrument 7, a line printer, but your program does not need the data, then you can use IBXFER.

```
CALL IBXFER (10, 5, 7)
```

This is the same as example 2 for IBRECV, except that the data is not stored in the array MESSAG.

You can also use:

```
CALL IBXFER (10, 'EG')
or
CALL IBXFER(10, 'E', 'G')
or
BYTE ADDR(3)
DATA ADDR /'E', 'G', 0/
        .
        .
        .
CALL IBXFER (10, ADDR)
or
BYTE ADDR(3)
DATA ADDR /5, 7, 0/
        .
        .
        .
CALL IBXFER (10, ADDR)
```

5.3.5 Data Receive Status (IBRCVS)

Use IBRCVS to obtain the status of a data transmission initiated by one of the following routines: IBRECV, IBARCV, IBXFER, IBAXFR. The driver terminates the receipt or transfer of data when one or more of three conditions is met:

- The length limit (as specified in bytes) is reached.

- A terminator character is detected.
- The EOI line is asserted.

The IBRCVS routine allows the user to determine whether termination actually occurred before completion of the data transmission. If an instrument is still in the process of transmission, the user can recall the receive or transfer routine that initiated the transmission to bring about proper completion. If termination was indeed premature, the instrument may have additional data to transmit, but the driver as controller will not recognize it. This condition will hang the bus.

Note that with termination of a receive or transfer function (excluding that initiated by IBFREE), the actual length of the transmission in bytes is also returned to the user (via IBWAIT if the transmission was asynchronous). This information may be instrumental in determining whether termination was premature.

IBRCVS can be called as a FORTRAN subroutine or function:

CALL IBRCVS ([iend])

or

iend = IBRCVS ( )

where:

iend is an integer variable that receives the receive-status code. There are 3 meaningful bits in iend as follows:

- Bit 0 (least significant bit) if set indicates that transmission was terminated upon reaching the user-specified byte length limit.
- Bit 1 if set indicates that transmission was terminated upon detection of a terminator character.
- Bit 2 if set indicates that transmission was terminated upon detection of the EOI line asserted.

The receive-status word is returned as a value such that $0 < iend < 8$, iend being one of all possible combinations of these three bits excluding the case in which all bits are clear. In this way, it can be determined which set of conditions existed upon termination of the data transfer.

For example, if iend = 1, we know that the length limit was reached, no terminator was detected, and EOI was not asserted. If iend = 7, the length limit was reached, a terminator character was detected, and the EOI line was asserted upon transmission termination.

5.4 Asynchronous Data Transmission

The asynchronous subroutines (see Section 1.9) IBASND, IBARCV, and IBAXFR are like their synchronous correspondents above, except that they return to the FORTRAN program in progress while the data is being transmitted. You must call IBWAIT or any other IB routine before using the transmitted data, to be sure that the transmission is complete.

These asynchronous routines cannot be used as functions. If IBWAIT is the first IB routine called after an asynchronous routine, it can be used as a function to return the number of bytes transmitted, just as IBRECV and IBXFER would (see IBWAIT, Section 10.4).

The synchronous functions IBSEND, IBRECV, and IBXFER automatically call IBWAIT before returning to the FORTRAN program in progress.

5.4.1 Asynchronous Send (IBASND)

CALL IBASND (msg-array [, [msg-len] [, lnrs]])

where:

msg-array	The message that you want to send in single quotes, or the name of the array in which that message is stored. If format conversion of a numerical value is required, use the FORTRAN statement ENCODE (see the *RT–11/RSTS/E FORTRAN IV User's Guide*)before sending the message.
msg-len	The number of bytes you want to send from the array. If msg-len is −1, the contents of the array are sent until a null byte (the integer 0) is reached. The character 0 terminates the transmission but is not sent. Specify −1 for msg-len, when you send a message in quotes, or when the last character in the message is followed by a null byte. If msg-len is not specified, the effect is the same as specifying −1. If msg-len is 0, no message is sent. The listeners specified in this routine replace those previously on the default listener list. This can be used to change the default listener list without sending a message. If msg-len is greater than 0, it specifies the number of characters that will be sent. The character 0 is sent like any other character. It does not terminate the message.
lnrs	The optional listener specification. If you do not specify this argument, IBASND uses the default listener list.

5.4.2 Asynchronous Receive (IBARCV)

CALL IBARCV (array, len-limit [, [tkr] [, lnrs]])

where:

array	The name of an array in which data received from the talker is stored. If format conversion to numeric values is required, use the FORTRAN statement DECODE (see the *RT–11/RSTS/E FORTRAN IV User's Guide*) after the data is received.
len-limit	A positive integer or 0 that specifies the maximum number of characters to be stored in the array. This integer should be no greater than the size of the array. You should specify len-limit to accommodate any anticipated number of characters. A 0 here means that no characters will be received. This argument must be specified. No default value is allowed.
tkr	The optional talker specification. If you do not specify a talker, IBARCV uses the default talker list.
lnrs	The optional listener specification. If you do not specify a listener, IBARCV does NOT use the default listener list. Instead, the IBV11–A/IB11 is the ONLY listener. Any default listeners are removed from the IEEE bus. The default listener list remains unchanged. IBRECV and IBARCV are the only routines in which the default listener list is not used if you do not specify a listener.

To use the current default listener list, specify −1. The default listener will listen, along with the IBV11–A/IB11.

To add new listeners to the current default listener list, specify −1, and then enter the addresses of the instruments you want to listen. This also applies for any other routine.

To specify listeners, do so without using −1. These new listeners become the default listener list, as in any other routine. These instruments and the IBV11–A/IB11 all listen.

5.4.3 Asynchronous Transfer (IBAXFR)

CALL IBAXFR ([len-limit] [, [tkr] [, lnrs]])

where:

len-limit	A positive integer (or 0) that specifies the maximum number of characters to be transferred. You should specify len-limit to accommodate any anticipated number of characters. A 0 here means that no characters will be transferred. Default or −1 allows the maximum number of characters (65,535) to be transferred.
tkr	The optional talker specification. If you do not specify a talker, IBAXFR uses the default talker list.
lnrs	The optional listener specification. If you do not specify a listener, IBAXFR uses the default listener list.

5.4.4 Free Transfer of Data (IBFREE)

IBFREE is similar to IBXFER. It transfers data from a talker to listeners, but is not monitored by the IB routines. The IBV11–A/IB11 neither talks nor listens. Data passes freely from talker to listener without the IB routines checking for terminating characters, EOI, or message length. This message transfer continues independent of your program, until you call IBUNT or any other IB routine that sends data or commands on the IEEE bus.

CALL IBFREE ([tkr [, lnrs]])

where:

tkr	The optional talker specification. If you do not specify a talker, IBFREE uses the default talker list.
lnrs	The optional listener specification. If you do not specify a listener, IBFREE uses the default listener list.

Chapter 6
Instrument Control Commands

6.1 System Commands

Most instruments on the IEEE bus can operate under either the local controls on their front panels, or under remote control of the IBV11–A/IB11. The first IB routine to communicate with a particular IBV11–A/IB11 unit causes the Remote line of the IEEE bus to be set. The IBV11–A/IB11 is ready to administer programmed instruction to the instruments in the system. An instrument becomes ready to receive instructions under remote control when it is first addressed.

6.1.1 Remote Disable (IBRDA)

If you wish to turn Remote off while running a program, use Remote Disable (IBRDA).

CALL IBRDA

IBRDA turns Remote off and returns to the program in progress after a 100 microsecond delay. Addresses in the default lists are removed from the IEEE bus, but the lists themselves remain intact. IBRDA delays 100 microseconds to ensure that any Go To Local command (see Section 6.3.1) will be undone if remote is enabled again.

6.1.2 Remote Enable (IBREN)

If you wish to return to remote control, use Remote Enable (IBREN).

CALL IBREN

IBREN turns Remote on and returns immediately to the program in progress.

6.1.3 Remote Status (IBREMO)

You can determine whether Remote is on or off by using the function IBREMO. When used with its optional argument n omitted, it returns an integer that reports the status of Remote. If this integer is 0, Remote is off; if the integer is -1, Remote is on.

IBREMO can also be used to turn Remote on and off at the same time it reports the previous status. If IBREMO turns Remote off, it waits 100 microseconds before returning to the program in progress, just as IBRDA does. Also like IBRDA, it removes the addresses in the default lists from the IEEE bus, but the lists themselves remain intact.

When n is specified, IBREMO either turns Remote on (any non-zero integer) or off (0) and then returns an integer value (0 or -1) indicating the previous status of Remote.

iold = IBREMO ([n])

where:

iold	An integer value returned by the function IBREMO, indicating the prior status of Remote. 0 is off. -1 is on.
n	An optional integer, zero or non-zero. A 0 turns Remote off and returns an integer to iold, indicating the prior status of Remote. A non-zero integer turns Remote on and returns an integer to iold, indicating the prior status of Remote. If n is omitted, the status of Remote is simply reported and remains unchanged.

Examples:

1.
```
        CALL IBREN            !Turns Remote on.
        I=IBREMO()            !Sets I equal to -1
        TYPE 10, IBREMO(0)    !Types -1, then turns Remote off.
        TYPE 10, IBREMO(-1)   !Types 0, then turns Remote on.
   10   FORMAT (1X, I2)
```

2.
```
        SUBROUTINE REMOTE
        I=IBREMO(-1)          !Turns Remote on
        IF (I.EQ.0) TYPE 10
   10   FORMAT (1X,'Remote was off and we just turned it on.')
        RETURN
        END
```

6.1.4 Interface Clear (IBIFC)

Each instrument communicates with the IEEE bus through an interface. You can clear all of the interfaces with the Interface Clear command.

CALL IBIFC

All instrument addresses are removed from the bus, so that no instrument is addressed to talk and no instrument is addressed to listen. The default lists remain intact, however, and any service routine in your program remains enabled.

This is an extreme measure, and should be used only to recover from unusual circumstances that prevent normal operations, such as a system hang up or a control conflict.

Calling IBIFC disables the Remote bus line and then enables it again. Special conditions such as Local Lockout (see Section 6.2.1) or Go To Local (see Section 6.3.1) are undone. Instruments in your system come under remote control again when they are readdressed by an IB routine.

6.2 Universal Commands

Universal commands apply to all instruments in the system. Each instrument capable of responding does so in a manner appropriate to its function. Because universal commands are sent to all instruments, specific addressing is not necessary. The default talker and listener lists are unaffected.

6.2.1 Local Lockout (IBLLO)

Many instruments that operate on the IEEE bus have a front panel LOCAL/RESET button. When this button is pressed, the operator can regain local control of an instrument that is under remote control of the IBV11–A/IB11. You can prevent such accidental return to local by calling (IBLLO).

CALL IBLLO

Now any instrument in your system with a LOCAL/RESET button ignores the position of that button and continues under the remote control of the IBV11–A/IB11.

To regain local control of bus instruments after calling Local Lockout, call Remote Disable (IBRDA).

CALL IBRDA

IBRDA may be followed by IBREN if remote control is desired when the local button is not pressed. All instruments are now ready for manual operation under local control as soon as their LOCAL/RESET button is pressed. Default lists are unaffected by IBLLO.

6.2.2 Device Clear (IBDCL)

Calling IBDCL clears all instruments on the IEEE bus. Each instrument returns to a predefined state appropriate to its function. See the operating manual of each instrument for a description of its predefined state.

CALL IBDCL

The default talker and listener lists are unaffected. The next IB routine requiring a talker or listener uses the pre-established default list(s) if no talker or listener is specified.

6.2.3 Untalk (IBUNT)

Normally when you specify a new talker in a routine requiring a talker, the former talker is automatically removed from the bus and the talker default list. Calling IBUNT removes the current talker from the IEEE bus only, without specifying a new talker.

CALL IBUNT

Now no talker is specifically addressed. However, the default talker list is unaffected. Any following IB routine requiring a talker uses the default talker list if no talker is specified.

6.2.4 Unlisten (IBUNL)

Calling IBUNL removes all addressed listeners from the IEEE bus.

CALL IBUNL

Now no instrument is on the IEEE bus as a listener. The default listener list is unaffected. Any following IB routine requiring a listener uses the default listener list if no listener is specified.

6.2.5 Parallel Poll Unconfigure (IBPPU)

The Parallel Poll Unconfigure (IBPPU) universal command prevents all instruments previously prepared to respond to a parallel poll by IBPPE (see Section 7.2.1) from responding. For a further description of this routine, see Section 7.2.3.

6.3 Addressed Commands

Addressed commands affect only instruments specifically addressed as listeners. The default listener list is used if the listener specification is omitted.

6.3.1 Go To Local (IBGTL)

When you are operating under remote control, it may be necessary to go to local control of a particular instrument to perform some function that cannot be done under remote control. You may simply want instruments that are not part of the current program to function locally. Use Go To Local (IBGTL), specifying instruments as listeners.

CALL IBGTL ([lnrs])

where:

lnrs	Optional listener address specification. If this argument is not specified, the default listener list is used.

After IBGTL is called, the instruments specified as listeners operate independently under local control.

Go to Local is effective even though IBLLO has been called. IBLLO disables the LOCAL/RESET button. Instruments specified as listeners in the IBGTL call operate under local control, but the LOCAL/RESET button remains disabled.

After you complete operation under local control, return to remote control by calling IBUNL, or any IB routine requiring a listener address, without defaulting the listener address.

Example:

Use IBGTL to return devices 9 and 14 to local (manual) control.

CALL IBGTL (9, 14)

6.3.2 Selected Device Clear (IBSDC)

It might be useful at times to clear selected instruments on the IEEE bus returning each to its own predefined state, generally the state when first powered on. IBSDC differs from IBDCL in that only those instruments specified as listeners are cleared.

CALL IBSDC ([lnrs])

where:

lnrs	Optional listener address specification. If this argument is not specified, the default listener list is used.

Example:

Use IBSDC to return instruments 6, 8, and, 24 to their predefined states.

CALL IBSDC (6, 8, 24)

6.3.3 Group Execute Trigger (IBGET)

You may want to execute the functions of more than one instrument in your system at the same time. Group Execute Trigger (IBGET) simultaneously triggers the functions of multiple instruments addressed as listeners. Not all instruments are designed to respond to IBGET. The operator's manual for each instrument tells whether that instrument responds, and what its response is. The response is instrument specific.

IBGET is often used with IBSEND to trigger the function of a single instrument. IBSEND sends programmed instruction to prepare the instrument to respond. IBGET then triggers the response.

CALL IBGET ([lnrs])

where:

lnrs	Optional listener address specification. If you do not specify this argument, the default listener list is used.

Examples:

1. Use IBGET to take simultaneous readings of a voltmeter (17), an ammeter (19), and a thermometer (4). These instruments have been prepared by appropriate IBSEND commands.

   CALL IBGET (17,19,4)

2. Example 3 of Section 5.3.3 shows the use of IBGET to trigger a single instrument.

6.4 Programmed Commands (IBCMD)

You can augment the existing lists of universal and addressed commands by using IBCMD. Each addressed command is associated with a decimal value in the range 0 through 15, and each universal command is associated with a decimal value in the range 16 through 31. (See Appendix B.) Some of these commands have specific meanings. Many of them are treated as separate IB routines (for example, IBLLO and IBGET). Several possible commands do not yet have a specific meaning associated with them. If such a meaning becomes defined for a command, IBCMD will allow you to use it, even though the existing IB routines have no separate mnemonics for it.

CALL IBCMD (n [, lnrs])

where:

n	The IB command specification. Enter a decimal value in the range 0 through 31. Specify addressed commands with values in the range 0 thru 15. Specify universal commands with values in the range 16 through 31. (See Appendix B.)
lnrs	Optional listener address specification. If you do not specify this argument, the default listener list is used. Listener specification applies to addressed commands only. You cannot specify listener addresses for a universal command.

Examples:

1. Assume that instrument 5 in your system recognizes a new addressed command with a code of 12 that causes it to send out to the bus the ASCII character string "general purpose instrument bus". Use IBCMD to execute this new addressed command.

   CALL IBCMD (12,5)

   Even if other instruments recognize this command they will not respond, because they are not addressed to do so.

2. Assume that several instruments in your system recognize a new universal command with a code of 29 that will ring a bell. Use IBCMD to execute this new universal command.

   ```
   CALL IBCMD (29)
   ```

   Although this is a universal command, it rings the bells of only those instruments that recognize its code 29.

Chapter 7
Checking Instrument Status

Instruments on the IEEE bus can report their status to the IBV11–A/IB11. Some instruments are designed to report their status by responding to a serial poll. Others report their status by responding to a parallel poll. Some can respond to both, but not simultaneously.

Serial polling checks the status of instruments one at a time. An instrument responds to the serial poll by sending its status byte. The status byte can report more than one piece of status information.

Parallel polling checks the status of one or more instruments simultaneously. Each instrument polled indicates one piece of information, its status bit. This status bit is different from the status byte used with serial polling.

7.1 Using Serial Polling to Check Instrument Status

When you serial poll an instrument on the IEEE bus, it sends a byte of status information. Bit 6 (octal value 100) of this status byte has a special meaning: it is set when the instrument is issuing a service request (see Chapter 8) and clear when it is not. The other bits have no reserved meaning, but indicate aspects of the current status of an instrument that are defined in the instrument manual.

Instruments request service from the IBV11–A/IB11 by asserting the service request (SRQ) bus line. However, since all instruments on the bus share this line, the IB routines cannot recognize which instrument is requesting service until a serial poll is performed. Once the instrument requesting service is identified, it stops asserting the SRQ bus line.

You can use either of two methods for identifying the device requesting service through the IBSRQ routine (see Section 8.1.3). One method is to allow the driver to initiate a serial poll once it detects assertion of the SRQ line of the IEEE bus. This method is referred to as a "driver-initiated" serial poll and entails the use of a user-written service routine. The second method is the 'user-initiated' serial poll. No service routine name is included in the IBSRQ call; however, delayed SRQ processing must be specified to enable the system to perform the serial poll successfully while the SRQ line is asserted.

The operating manual for each instrument tells whether the instrument is designed to respond to a serial poll, and what that response indicates. An instrument that can request service must be able to respond to a serial poll.

7.1.1 Determining an Instrument's Status (IBSTS)

Use the function IBSTS to check the status of a single instrument by performing a serial poll on that instrument.

istat = IBSTS ([tkr])

where:

istat	The integer variable that receives the status byte. Its value is in the range 0 through 255 (octal values 0 through 377). If bit 6 (octal value 100) is set, the instrument addressed is requesting service. It will stop requesting service because of this serial poll.
tkr	The address specification of the device being polled. If no address is specified, IBSTS uses the default talker list.

Example:

Assume that instrument 8 is a plotter which indicates in bit 5 (octal value 40) of its status byte whether its pen is up or down.

```
      IPEN = IBSTS(8)
      IF((IPEN.AND."40).NE.0) TYPE 100
      IF((IPEN.AND."40).EQ.0) TYPE 101
100   FORMAT (1X,'My pen is up.')
101   FORMAT (1X,'My pen is down.')
```

7.1.2 Determine the Source of a Service Request (IBSPL)

You can determine whether any instrument is requesting service by serially polling the group with the function IBSPL. The instruments are polled individually in the order in which they appear in the multiple talker specification. The polling continues until an instrument requesting service is found or until the list of specified talkers is exhausted. When an instrument requests service, an integer representing its status byte is returned to the variable istat. This is the same status byte as that returned by IBSTS. Another integer is returned to the variable index identifying the instrument requesting service (1 indicates the first listed, 3 indicates the third, and so forth). IBSPL returns to your program without polling any of the talkers listed after the one requesting service. If the complete list of talkers is polled and none is requesting service, the variable index contains 0 and the variable istat contains the status byte of the last device in the list.

index = IBSPL ([istat] [, tkr [, tkr [, ...]]])

where:

index	An integer variable containing the serial number of the first instrument requesting service. The integer 1 indicates the first instrument in the list, 2 indicates the second, and so forth. The integer 0 indicates that no instrument in the list is requesting service.
istat	An optional integer variable that receives the status byte of the last talker polled. The values are in the range 0 through 255 (octal 0 through 377).
tkr	The address specification of talkers being polled. More than one primary address can be specified, with optional secondary addresses as appropriate. Only the first talker specification may be defaulted. When IBSPL is done, the default talker is the last instrument polled.

This call is useful for serial polling just one instrument, since index indicates whether the instrument is requesting service. Therefore, it is not necessary to check bit 6 of istat.

Examples:

1. Take a serial poll of devices 1, 7, and 4.

```
        BYTE INSTR (4)
        DATA INSTR /1, 7, 4, 0/
        INDEX=IBSPL (ISTAT, INSTR)
        IF (INDEX.EQ.0) TYPE 1001
        IF (INDEX.NE.0) TYPE 1002, INSTR(INDEX), ISTAT
1001    FORMAT(/1X,'None of the instruments is requesting service')
1002    FORMAT(/1X,'Instrument' ,I2, 'is requesting service'/
        1 /1X, 'Its status byte is', O4)
```

2. Serial poll instruments 4 and 7.

```
        GO TO (100, 200, 300), IBSPL(ISTAT, 4, 7)+1
100     TYPE 101
101     FORMAT (/1X,'No one requesting service')
          .
          .
          .
200     TYPE 201
201     FORMAT (/1X,'Instrument 4 is requesting service')
C       WE MIGHT WANT TO RECEIVE DATA FROM
C       INSTRUMENT 4 AT THIS POINT
          .
          .
```

```
             .
300     TYPE 301
301     FORMAT (/1X,'Instrument 7 is requesting service')
             .
             .
             .
```

3. Is instrument 3 requesting service?

```
        IF (IBSPL(ISTAT, 3) .NE.0) TYPE 1001
1001    FORMAT (/1X,'Instrument 3 is requesting service')
```

7.2 Parallel Polling

Parallel polling provides a quick means of determining information concerning the status of more than one instrument. This is because several instruments can be polled simultaneously. An instrument indicates a single piece of information with its status bit. Do not confuse this status bit with the status byte used in serial polling. The status bit is either set or clear to indicate one piece of information concerning an instrument's status. An instrument capable of responding to a parallel poll may have to be enabled beforehand to do so with IBPPE. The operating manual for each instrument tells whether the instrument is designed to respond to a parallel poll. The response can be unique for as many as eight instruments.

7.2.1 Parallel Poll Enable (IBPPE)

Some instruments in your system can respond to parallel polling, but you may have to prepare them to do this. Use parallel poll enable (IBPPE) to assign a data line on which the instrument can respond, and a sense (0 or 1) to indicate whether the instrument is to set that data line during parallel poll when its status bit is set (1) or when it is clear (0). IBPPE only tells instruments how to respond to a parallel poll; it does not actually conduct a parallel poll. This is done by IBPPL.

CALL IBPPE (isense, line [, lnrs])

where:

isense	An unsigned integer that specifies what state of the status bit will cause the data line set when a parallel poll is conducted by calling IBPPL. If isense is set to 0, the instrument sets the data line in response to a parallel poll if its status bit is 0 (when it is not requesting service). If it is set to any other value, the instrument sets the data line when it is requesting service.
line	Data line assignment. This tells the addressed instrument(s) which data line to respond on when a parallel poll is conducted. It is an unsigned integer in the range 1 through 8, corresponding to the 8 data lines on the IEEE bus. Data line 1 indicates the least significant bit of the data byte (see Section 7.2.4).

lnrs Optional listener specification. This argument specifies the instrument(s) to which the data line and its sense are assigned. If no listener is specified, the default listener list is used.

Usually, the best practice is to assign a separate data line to each instrument. If this is done, only one instrument can assert any one data line during a parallel poll.

7.2.2 Parallel Poll Disable (IBPPD)

Parallel poll disable (IBPPD) selectively disables instruments addressed as listeners from responding to a parallel poll. It clears the line and sense assignments specified to instruments with IBPPE. This is like IBPPU, except that it applies only to the instruments addressed as listeners.

CALL IBPPD ([lnrs])

where:

lnrs Optional listener specification. If no listener is specified, the default list is used.

7.2.3 Parallel Poll Unconfigure (IBPPU)

Calling IBPPU stops all instruments previously addressed as listeners in a call to IBPPE (see Section 7.2.1) from responding to a parallel poll. Compare this with IBPPD, Section 7.2.2

CALL IBPPU

Now no instrument can respond to a parallel poll until IBPPE is called again. Calling IBPPU does not alter the default listener list.

7.2.4 Parallel Polling (IBPPL)

Use IBPPL to determine the state of instruments on the IEEE bus. Only instruments that can respond and are enabled by IBPPE will respond. A single piece of information concerning the status of an instrument is recognized by whether a data line assigned to the instrument is asserted. Each data line has an associated integer value. When an instrument asserts its data line during a parallel poll, IBPPL returns the associated value to the integer variable ireslt. If only one data line is asserted, its associated value is recognized easily. If more than one data line is asserted, ireslt contains the arithmetic sum of the values.

ireslt = IBPPL ([n1 [, ..., n8]])

where:

n1, ..., n8 A list of up to 8 integers in the range 1 through 8 corresponding to the 8 data lines. Only data lines specified here are included in the poll. If no lines are specified, all 8 lines are included in the poll.

ireslt An integer variable that receives the response to the poll. Each data line has an associated integer value. Ireslt is set equal to the sum of the integers associated with data lines that are both asserted and included in the poll. The result of the poll contained in ireslt is a value in the range 0 through 255.

The following list shows the contribution of each included and asserted data line to the total result. If more than one device asserts the same data line, the associated value of that data line is added only once to the total result.

Data Line	Value Added To Ireslt
1	1
2	2
3	4
4	8
5	16
6	32
7	64
8	128

Chapter 8
Service Requests

8.1 Using Service Requests

Some instruments on the IEEE bus are capable of requesting service from the IBV11–A/IB11 Instrument Bus Interface. The operating manual for each instrument tells whether the instrument can request service and, if so, for what purpose.

Service requests are indicated on the SRQ interface control line of the IEEE bus. The IBV11–A/IB11 can check the line to determine whether a service request is present. All instruments on the bus, however, share this line. If SRQ has been asserted, the IB routines cannot determine which instrument is involved until a serial poll is performed. The service request can only be cleared by performing a serial poll.

You can specify to the driver one of two methods of identifying the device requesting service with the IBSRQ routine. One method is to allow the driver to initiate a serial poll once it detects assertion of the SRQ line of the IEEE bus. This method is referred to as a "driver-initiated" serial poll. The second method is the 'user-initiated' serial poll. Here it is the user's responsibility to detect assertion of the SRQ line and to perform the serial poll.

Upon performing a serial poll and identifying the instrument requesting service, you can also examine the instrument's status byte to determine what action is appropriate. If you have chosen the driver-initiated serial poll, this examination and action takes place in a user-written service routine to which the system transfers control immediately following the serial poll. This service routine executes asynchronously with respect to the rest of your program. If you have chosen the user-initiated serial poll method, you will examine the status byte and perform the requested service synchronously with respect to the rest of the program.

You must call IBSRQ to be able to use either the driver–initiated or the user–initiated method. Specifying both the service routine name and the status byte variable constitutes selection of the driver–initiated approach. Alternately, defaulting the service routine name and at the same time specifying delayed processing, designates a user–initiated serial poll.

Several considerations are essential to a clear understanding of the driver's role in handling service requests with driver–initiated serial polls. These considerations include:

- the need to provide the driver with a list of instruments capable of requesting service
- the information the driver passes to the service subroutine
- the situations that generate error conditions
- the consequence of more than one instrument requesting service at one time

These four matters are expanded upon below.

Your program must supply a list of all instruments connected to the IBV11–A/IB11 unit that are capable of generating a service request. This list determines which instruments will be serially polled by the driver. IBDEV permits you to specify this list.

After the requesting instrument has been identified, the driver returns its bus address(es) and status byte to variables in the service routine which you have specified in the IBSRQ call. These variables should be placed in a COMMON statement, because their specification occurs in a part of your FORTRAN program that is independent of the service routine.

If you fail to call IBDEV with a device list, or if you omit from this list a device which is indeed capable of asserting the SRQ bus line, it is possible that the driver will detect the SRQ asserted condition but be unable to identify the device requesting service. This is an error condition, but one which is handled in a manner independent of the error processing scheme presented in Chapter 9. In particular, this is called a locator error and is identified as a −1 in the status byte variable. Therefore, to check for this error condition, the first step your service routine should take is to test the status byte for a −1. If −1 is detected, the service routine should be immediately exited.

There are three other error codes: –2, –3, and –4. An error code of –2 means that a timeout error occurred during a serial poll, –3 means that there was a conflict over control of the bus, and –4 means that there was no listener on the bus.

Note that the status byte variable normally returns an 8–bit byte in the least significant byte of the 16–bit variable. The upper eight bits are therefore always zero, except during an error condition. When an error occurs, the full 16 bits are used to return a negative number. Therefore, there can be no confusion between the error codes and the normal 8–bit status bytes.

Since all instruments use the same IEEE bus line to request service, the SRQ line may still be asserted following the identification and complete servicing of one instrument. The driver will continue to initiate serial polls in this case, providing complete service for each instrument until all requesting devices have been serviced and the SRQ line is clear. Complete service for each requesting instrument implies the execution of the user-written service routine.

8.1.1 Testing the Service Request Flag (IBSRQF)

Use the function IBSRQF to determine if any instrument on the IEEE bus is requesting service.

iflag = IBSRQF ( )

where:

iflag An integer variable that indicates if any instrument is requesting service. If no instrument is requesting service, the value 0 is returned to iflag. If one or more instruments are requesting service, the value -1 is returned to iflag. (See the *IB11 UNIBUS to IEEE Instrument Bus Interface Installation Manual.*)

You can also call IBSRQF as a subroutine.

CALL IBSRQF (iflag)

Example:

Your system contains only one instrument (3) that can request service. When you press its CALL button, it requests service, indicating that it has data to send.

```
    BYTE VALUE (15)
    .
    .
    .
    TYPE 10
10  FORMAT (1X,'Press CALL button to take a reading.')
20  IF (IBSRQF().EQ.0) Go to 20    !Wait for SRQ
    CALL IBSTS(3)    !Serial poll tells 3 to stop requesting
                     !service
    CALL IBRECV (VALUE,15,3)    !Get data
    .
    .
    .
```

8.1.2 Specifying the Service Request Device List (IBDEV)

IBDEV supplies the list of all instrument bus addresses which are capable of generating service requests. When a driver-initiated serial poll is selected and an SRQ request occurs, the driver polls the devices on this list to determine which instrument caused the request. Should the SRQ line be set before such a list is made available, or should the SRQ line be set by a device not included in the device list, an error value of -1 will be returned in the status byte variable which is specified in the IBSRQ call.

CALL IBDEV (dev–1st)

dev–1st	A list of primary and/or secondary bus addresses of instruments that are capable of generating an SRQ request. The maximum number of devices that may be specified is 30 (decimal). At least one address specification is required. Byte array addressing is preferable with specification of a large number of devices. The order in which devices are listed constitutes a priority structure with respect to servicing if multiple service requests occur. Thus, the first entry is serviced first and so on.

Example: see example in Section 8.1.3.

8.1.3 Specifying the Service Request Routine (IBSRQ)

IBSRQ specifies information related to the disposition of service requests. This subroutine call selects either the driver–initiated or the user–initiated serial poll method of handling service requests.

If you minimally specify the subroutine–name and status byte variable, the system will execute the driver–initiated serial poll. In this case, specification of the remaining variables is an important consideration with respect to the associated service routine. Your service routine will use the bus address variables and the status byte to recognize the device which requested service and the nature of the service required. Additionally, the dev–1st parameter will notify the driver of precisely which devices on the serial poll list are to be dealt with by the service routine. Note that an instrument which is included in the list supplied to IBDEV need not necessarily be handled by the service routine. If this is the case, the instrument(s) in question will have their requests recognized and cleared by the driver–initiated serial poll. Synchronous processing by your main program will continue, and no attempt will be made to call a service routine.

On the other hand, if you default the subroutine–name and specify the idelay parameter to indicate delayed processing, you designate the user–initiated serial poll. These two steps are prerequisites to the calling of IBSRQF and /or one of the serial poll routines.

CALL IBSRQ ([subroutine–name],[ipaddr],[isaddr],[istat],[idelay],[dev–lst])

where:

subroutine–name	is the name of the user–written subroutine that will handle service request interrupts generated by instruments included in the dev–1st. If no routine is specified, the current subroutine assignment is disabled; this allows you to conduct your own serial poll (as long as idelay = 1 for delayed processing). If this argument is defaulted, the only other significant parameter specification is the idelay argument.

ipaddr	is an integer variable to receive the primary bus address of the instrument that asserted the SRQ line. This variable is used by a service routine which services more than one device to identify which instrument issued the request currently being serviced.
isaddr	is an integer variable to receive the secondary bus address of the instrument that asserted the SRQ line. If the instrument requesting service does not have a secondary address, a value of -1 will be returned. This variable is used by a service routine which services more than one device to identify which instrument issued the request currently being serviced.
istat	is an integer variable to receive the status byte returned by the requesting device in response to the driver's serial poll. This variable is used by a service routine to determine the nature of the current request and must be specified if the subroutine–name parameter is specified. The error values are also reported in this variable as described in Section 8.1.
idelay	is a flag determining whether SRQ processing is to take place as soon as the service request is received, or is to be delayed. If idelay = 0, the system will abort any transfer that is in progress at the moment of the SRQ interrupt and the SRQ processing will take place immediately. If idelay = 1, the SRQ processing will be delayed until the end of the current transaction or a device timeout, whichever occurs first. If idelay is defaulted, the abort case will be assumed. If the SRQ subroutine–name is defaulted, this flag should be set to 1 to prevent any SRQ interrupts from aborting the current transaction and to allow the user–initiated serial poll to be performed.
dev–lst	is a list of primary and/or secondary bus addresses of instruments to be serviced by the user–specified routine. This list is a subset of the IBDEV list. The maximum number of addresses is 30 (decimal). If this argument is defaulted, service requests from all of the instruments listed in the call to IBDEV will be directed to the user's serivce routine.

Requirements of the user–written service routine:

1. The service routine can have no arguments and can only communicate with the main program that has been interrupted by means of COMMON areas. The variables which will receive the primary address, secondary address and status byte information will often be part of this COMMON area. This procedure allows specification in the IBSRQ call and provides access in the service routine.

secondary address and status byte information will often be part of this COMMON area. This procedure allows specification in the IBSRQ call and provides access in the service routine.

2. The service routine can change terminators, but when the service routine is completed and control returns to the main program, the terminators in the main program remain as they were changed by the service routine. The same applies to the timeout value.

3. The main program and the service routine establish and maintain separate default talker and listener lists. An error occurs if talker or listener specifications are not given in a routine in which they are required before their respective default lists have been established.

4. If the user-written service routine attempts to call IBDEV, IBSRQ, IBSTS, or IBSPL, an error condition results.

5. User-written subroutines return through the normal RETURN statement.

6. In Extended Memory (XM) systems, user-written routines must reside in the root segment in low memory (below 28K words) and the associated job must be privileged.

7. In Extended Memory (XM) systems, the user-written service routine and any data areas used or modified by the user-written service routine must not reside in the physical memory mapped by kernel page-address-register number 1; this encompasses the virtual address range from 020000 (octal) to 037776 (octal). If necessary, use the LINKER program to enforce this restriction. (For additional information, see the RT–11 Version 4 System User's Guide.) Additionally, the routine that establishes the user-written service routine (by calling IBSRQ) and the user-written service routine itself must employ privileged, not virtual, mapping.

Example:

Assume that instrument 4 is a voltmeter which can be programmed to automatically trigger a reading every 0.1 seconds and to assert the SRQ bus line when it is ready to transfer data. Assume that this capability can be programmed with the ASCII string 'PROGRAM'. The following sample illustrates the way in which IBSRQ and IBDEV would be used to handle the service requests.

```
      COMMON ISTAT
      EXTERNAL IBSERV
      .
      .
      .
      CALL IBDEV (4,5,1,7)
      CALL IBSRQ (IBSERV,,,ISTAT,IDELAY,4)
      CALL IBUNT
      CALL IBUNL
      CALL IBDCL
      CALL IBSEOI ('PROGRAM',,4)
      CALL IBUNL
      PAUSE 'DATA ACQUISITION PHASE'
      IF ((ISTAT.EQ.-1).OR.(ISTAT.EQ.-2)) GO TO 500
      .
      .
      .
500   STOP 'END'
      END

      SUBROUTINE IBSERV
      COMMON ISTAT
      DIMENSION MESSAG (10)
      .
      .
      .
      IF (ISTAT.EQ.-1) GO TO 900
      IF (ISTAT.EQ.-2) GO TO 910
      LEN = IBRECV (MESSAG,10,4)
      CALL IBRCVS (IEND)
      CALL IBUNT
      TYPE 50, IEND,(ITXTBF(I),I=1,LEN)
50    FORMAT(' The receive status word is: ',I1/
      1,' The message received from the voltmeter was: ',10A2)
      .
      .
      .
      GO TO 950
C
900   TYPE 901
901   FORMAT(' Locator error.')
      GO TO 950
910   TYPE 911
911   FORMAT(' Timeout error.')
950   RETURN
      END
```

Chapter 9
Detecting and Reporting Errors

9.1 Errors and Error Messages

Each IB routine can detect and report errors to an internal error handling routine. This routine prints an error message to identify the problem and then returns control to the system monitor. This is called a fatal error because the program is not completed (see Section 9.1.1).

There are two forms of error messages: long and short. The long form explains more but requires more memory. The short form is an integer associated with each error type. You will receive long error messages unless you request short ones. If you want to use short error messages, see Section 4.5.

In the long form, the error message is preceded by "Error in routine xxxxxx:". For example:

Error in routine IBSEND: Invalid parameter

The corresponding short error message is

IB Error #3

Table 9–1 lists the error messages, the associated integer value for each error type, and the appropriate action to correct the error condition.

Table 9–1: IB Routine Error Messages

Error Type	Error Message	Action
1	Timeout	The IB routine waited for the next character to be sent or received for a longer time than specified in the IBTIMO routine. Specify a longer timeout (see Section 10.4), or check that all your instruments are connected, turned on, and correctly addressed (see Section 5.2).
2	Invalid instrument address	Primary addresses specified must be in the range 1 through 30, or ASCII characters '@ through '^' or 'SP' through '>', and secondary addresses must be in the range 200 through 230, or ASCII characters '' through '~'. (See Section 5.2).
3	Invalid parameter	Check that all parameters are valid. No parameter that must be specified can be defaulted. Each parameter must be within the limits defined in the description of the routine for which it is specified. All parameters must be of the proper data type. For example, integers must be used, not real numbers.
4	No default talker list available	The first IB routine in the main program, or in an SRQ routine, that required a talker specification cannot use the default talker list until a talker has been specified, to establish the default talker list.
5	No default listener list available	The first IB routine in the main program, or in an SRQ routine, that requires a listener specification cannot use the default listener list until listeners have been specified, to establish the default listener list.
6	SRQ abort of in progress transmission	The current bus transaction was aborted as a result of an SRQ interrupt (assertion of the SRQ line causes the interrupt). This is the default case. To allow completion of the transaction, specify the IBSRQ idelay argument equal to 1 (see Section 8.1.3).

Table 9–1: IB Routine Error Messages (Cont.)

Error Type	Error Message	Action
7	Invalid IBV11–A unit number	Rebuild IB.SYS after editing IBLOC.ASM to include information about the IBV11–A/IB11 unit specified (See Appendix C), or make corrections to the unit number specified in the call to IBUNIT.
8	IB.SYS is not loaded	Type: LOAD IB (see Section 4.3.6.) after the monitor prompt, then rerun your program.
9	Default talker list overflow	Default talkers exceed the capacity of the default lists. The default talker list can include a maximum of one primary and one secondary address. The length of this list cannot be extended.
10	Default listener list overflow	Default listeners exceed the capacity of the default listener list. The default listener list can include a maximum of 30 decimal listener address values. The length of this list cannot be extended.
11	Device issuing SRQ was not listed in a call to IBDEV.	Supply the complete list of devices capable of generating an SRQ in a call to IBDEV (see Section 8.1.2).
12	The user-written service routine attempted to call IBSRQ, IBDEV, IBSTS, or IBSPL.	Remove the call to IBSRQ, IBDEV, IBSTS, or IBSPL from the service routine.
13	Same instrument addressed to talk and listen	In calls to IBRECV, IBARCV, IBXFER, and IBAXFR, make sure that no instrument is specified as both talker and listener. If this error is set to be nonfatal, the instrument is addressed first as a listener, then as a talker, and transmission is allowed.
14	Conflict over control of the bus	Only one IBV11–A/IB11 must be in control of the IEEE bus at any one time. Make sure that no other instrument is attempting to control the bus.
15	No valid listener on the bus	Ensure that all IEEE bus cables are connected, and that all instruments are turned on and correctly addressed.

Table 9–1: IB Routine Error Messages (Cont.)

Error Type	Error Message	Action
16	No timeout support available.	Perform a sysgen on the system monitor in order to include support for device timeout, then rebuild the device driver and IBS subroutine library as outlined in Appendix C.

There is one known situation in which the IB routines do not detect an error condition. If your FORTRAN program is complex, it can use the stack heavily. If too many words are entered on the stack, the IBV11–A's vectors can be written into by the stack. When the next IB interrupt occurs, the vector location will contain incorrect values. This will generally cause a fatal error with no IB error message. The exact symptoms are unpredictable (possibly an "illegal instruction" message or an "illegal memory reference" message).

The solution is to relink your FORTRAN program using the BOTTOM switch to specify that the stack begin at an address greater than the default of 1000 (octal). For example, to allow 200 (octal) more bytes for the stack (see the *RT–11 System User's Guide*), type:

```
.LINK TEST,IBLIB/BOTTOM:1200
```

9.1.1 Setting Error Handling Characteristics (IBSTER)

The internal error handler should be adequate for most operations. However, you may need to modify the way it responds to certain errors.

Although this error handler cannot be called directly from your FORTRAN program, you can set parameters within it to define its response when an error is detected.

Each error type is identified by number in Table 9–1. By using IBSTER with a number identifying the error type, and another integer, you can define the response of the error handler to an error condition.

```
CALL IBSTER (nerr, ncount)
```

where:

nerr	An integer corresponding to the type of error for which you are setting the error handling characteristics (see Table 9–1).
ncount	An integer value that indicates the action to take to handle this type of error.

Value	Action
0	Prints the error message and returns to your program.
1	Prints the error message and exits to the monitor. This is the default case, when IBSTER has not been called for an error type.
2–127	Prints the error message and returns to your program. The value is decremented. You can use this feature to specify how many times you will allow this type of error to occur before it becomes fatal. If ncount falls in this range, the error is fatal when it occurs for the ncount'th time.
<0 or >127	Any value less than 0 or greater than 127 causes this error type to return to your program without printing the error message.

Examples:

After the following call is executed all future 'No default listener list available' errors will print a message and then return to your program. After the call, error 5 will never be fatal.

CALL IBSTER (5, 0)

After the following call each of the next three timeout errors prints a message and then returns to your program. The fourth timeout error will be fatal and the program will print a message and exit to the monitor.

CALL IBSTER (1, 4)

9.1.2 Reading the Error Flag (IBERRF)

IBERRF returns an integer that designates all error types that have occurred since the last call to IBERRF (or since the beginning of the program, if this is the first call to IBERRF). IBERRF returns to the integer variable maperr the arithmetic sum of the values associated with the different types of errors which have occurred, and which are listed in the call to IBERRF. Only the error types that are reported in maperr are cleared from the internal error log. For example, if you first call IBERRF with only error type 3 specified, no other errors will be reported at that time, even if they have occurred. If later you call IBERRF to report all errors (with no arguments specified), then it will report error 3 only if another error 3 has occurred since the previous call to IBERRF. However, it will report other errors whether they occurred before or after the previous call.

maperr = IBERRF ([n1 [, ..., n16]])

where:

maperr	The sum of the associated values of all error types that have occurred since the last call to IBERRF, or since the beginning of the program.
n1, ..., n16	A list of integers corresponding to the error types. The values must be in the range 1 through 16. Only the associated values of error types specified will be returned in the result to maperr. If no values are listed, all error types are included. The associated values of error types are listed in Table 9–2.

Table 9–2: Value Added to the Result in MAPERR

Error Type	Decimal	Octal
1	1	1
2	2	2
3	4	4
4	8	10
5	16	20
6	32	40
7	64	100
8	128	200
9	256	400
10	512	1000
11	1024	2000
12	2048	4000
13	4096	10000
14	8192	20000
15	16384	40000
16	32768*	100000

*You must check this value using the octal representation; the number is too large to represent as a FORTRAN decimal integer.

Example:

The following example checks to see if there is a valid listener on the bus. If there is none, the 'No valid listener' error condition exists, and its associated octal value 40000 is returned to the integer variable maperr. Other error conditions can also exist, but are not included. These are not cleared from the internal error flag, and can be detected in later calls to IBERRF. You can check for a valid listener much more easily by using IBLNR, as seen in the example in Section 10.6.

```
    BYTE MESSAG(30)
    CALL IBSTER(15, -1)          ! Make error 15 nonfatal, do
                                 ! not print message.
10  CALL IBSEND ('SETUP', ,7)    ! Try to send it a message.
    IF(IBERRF(15).EQ.0) GO TO 20 ! Instrument is on the bus,
                                 ! continue.

    PAUSE 'Turn on the equipment!'
    GO TO 10
20  CALL IBSTER(15, 1)           ! Make error 15 fatal again.
    CALL IBRECV(MESSAG, 30, 7)
             .
             .
             .
```

Chapter 10
Advanced Programming Techniques

10.1 Selecting an IBV11–A/IB11 Unit (IBUNIT)

An instrumentation system with more than one IEEE bus must have a separate IBV11–A/IB11 unit for each bus. The software can communicate through only one unit at a time.

The default control/status register address (defined by the symbol IB$CSR) and the default vector address (defined by the symbol IB$VEC) for the first IBV11–A/IB11 unit must be entered into the file IBLOC.ASM. The symbol IBNUNI in the file IBLOC.ASM must be set equal to the total number of IBV11–A/IB11 units in your system (from one to eight). Then the IB device driver must be re-built to accommodate all units in the system (see Appendix C). If these conditions are not met, or if the locations specified in the file IBLOC.ASM are not actually present on the computer, an error is generated and you cannot change from one IBV11–A/IB11 unit to another.

When the first IB routine is called in your program communication is automatically established with unit 0, unless that routine is IBUNIT specifying a different unit number. To change operations from one instrument bus to another, specify which IBV11–A/IB11 unit you want to communicate through by using IBUNIT either as a subroutine or as a function.

Each call to change units creates the need to establish new default talker and listener lists. The IB software maintains the following information separately for each unit: SRQ processing information, terminator lists, and timeout values.

SRQ processing can be active on only one unit at a time. The SRQ routine will remain active on that unit, no matter which unit has been selected by IBUNIT.

CALL IBUNIT ([nunit])

or

nold = IBUNIT ([nunit])

where:

nunit	An integer in the range 0 to 7 specifying the number of the new IBV11–A/IB11 unit. If this integer is omitted, the unit number remains unchanged.
nold	An integer variable that is set equal to the number of the IBV11–A/IB11 unit before this call to IBUNIT.

Example:

1. A system is specified to operate on unit 2. Then operation is changed to unit 1. A record is kept of the number of the unit (2) under which the system was formerly operating. The system is later returned to unit 2.

```
CALL IBUNIT (2)
    .
    .
    .
nold = IBUNIT (1)
    .
    .
    .
CALL IBUNIT (nold)
```

10.2 Defining Data Message Terminators (IBTERM)

Data messages are most commonly terminated by line feed LF or carriage return RET. However, this is not universal among all laboratory instruments. The operator's manual for each instrument tells which terminator(s) that instrument sends or receives. IBTERM allows you to specify which characters will be recognized as message terminators in future calls to IBRECV, IBARCV, IBXFER or IBAXFR. Message terminators are stored in the array you specify as the first argument in a call to IBRECV or IBARCV.

Before you call IBTERM, only LF and RET are recognized as terminating characters. This means that if a talker other than the IBV11–A/IB11 sends either LF or RET, the IBV11–A/IB11 stops the instrument from transmitting any further data at that point. In case your instrument sends message terminators of RET followed by LF with the EOI line asserted, be sure to define only LF as a valid terminator; otherwise, the RET is considered the end of the message, and the bus may hang due to no listener present to receive the LF.

Use IBTERM to change the message terminators from LF and RET to whatever terminators the various instruments in your system send. You can specify up to four message terminators at one time. When you do this, you establish a new list of terminating characters. RET and LF are no longer recognized as terminators unless they are included in the new list.

When no special message terminators are desired, remove them by calling IBTERM with no argument specified. LF and RET will not be restored automatically. You must call IBTERM with LF and RET specified to re-establish them as active message terminators.

Call IBTERM as a subroutine.

CALL IBTERM ([chr-val [, chr-val, ...]])

where:

chr-val	A decimal or octal integer value of the ASCII character being defined as a terminator, or a quoted ASCII character. Specify up to four values at one time. Designate a value as octal by preceding it with a double quote. When no argument is specified, IB routines recognize no message terminators.

A select list of ASCII characters commonly used as message terminators is shown below with their decimal and octal values. Any ASCII character with a decimal value in the range 0 through 127 (octal 0 through 177) can be used as a terminator. Appendix A contains a complete list of ASCII characters with their decimal and octal values.

Symbol	Decimal	Octal	Name
<ETX>	3	3	End of text
<EOT>	4	4	End of transmission
<LF>	10	12	Line feed
<VT>	11	13	Vertical tab
<FF>	12	14	Form feed
<RET>	13	15	Carriage return
<ETB>	23	27	End of transmission block
<EM>	25	31	End of medium
<SUB>	26	32	End of file
<ESC>	27	33	Escape
<FS>	28	34	File separator
<GS>	29	35	Group separator
<RS>	30	36	Record separator

Example:

If one of your instruments recognizes carriage return RET as a message terminator, and another recognizes form feed FF, you can specify these characters as message terminators with the following:

CALL IBTERM("15, "14)

or

CALL IBTERM(13, 12)

LF is no longer an active message terminator. RET is retained as a message terminator. When any instrument sends FF as a terminator, data transmission stops.

You can retain LF as an active terminator by including its integer value among those specified in the argument.

CALL IBTERM ("15, "14, "12)

10.3 Setting the Timeout (IBTIMO)

Use IBTIMO to specify how long the software will wait for an action to complete during data transmission. If, for any character in a message transfer, the addressed talker does not complete transmission within the specified timeout period, or the addressed listeners do not respond, a timeout error occurs.

As a prerequisite to specification of the timeout period, device timeout support must be included in your system monitor. Otherwise, this routine will return an error message. Additionally, if no timeout period is specified, or if the current one is disabled, the IB routines wait indefinitely for transmission to complete. This will cause the system to hang if for some reason the action never occurs. You can use IBTIMO as either a subroutine or as a function.

CALL IBTIMO ([nticks])

or

nold = IBTIMO ([nticks])

where:

nticks	An unsigned integer that specifies the length of the timeout period in ticks (65,535 maximum – see Appendix F for handling integers larger than 32,767). A tick is 1/60 second on 60Hz power and 1/50 second on 50Hz power. If some timeout period is specified, the system must have a system line clock. If nticks is specified as 0, the timeout feature is disabled. If nticks is defaulted, the current timeout setting remains unchanged.
nold	An integer variable whose value is the length in ticks of the previous timeout period before this IBTIMO was called.

Example:

The following example establishes an initial timeout period of one second (assuming 60 Hz line frequency). The timeout is then increased to three seconds, saving the initial timeout period in the integer variable nold. The initial timeout period is finally restored.

```
CALL IBTIMO (60)
        .
        .
        .
NOLD = IBTIMO (180)
        .
        .
        .
CALL IBTIMO (NOLD)
```

10.4 Waiting for Asynchronous Transmission to Complete (IBWAIT)

When you use IBASND, IBARCV, or IBAXFR, data transmission proceeds asynchronously while your program continues. IBWAIT waits for that data transmission to complete. It is called automatically at the beginning of every other IB routine.

```
CALL IBWAIT
```

If the last IB routine called before IBWAIT was IBARCV or IBAXFR, you can use IBWAIT as a function. Used this way, IBWAIT will return to the integer variable nbytes the number of bytes actually received by IBARCV or the number of bytes actually transferred during IBAXFR (just as IBRECV and IBXFER work as functions).

```
nbytes = IBWAIT ()
```

where:

nbytes	An integer variable that receives the number of character bytes transmitted.

10.5 Checking for Valid Listeners (IBLNR)

To ensure that instruments you want to be listeners are currently connected to the IEEE bus and turned on, use the function IBLNR. If any of the instruments specified as listeners are valid, IBLNR returns –1 (true) to the integer variable ivalid. If none of the instruments specified is currently a valid listener, the 'No valid listener' error condition exists, but does not cause an error or affect the error flag. IBLNR returns 0 (false) to the integer variable ivalid. This is especially useful in checking a single device.

ivalid = IBLNR ([lnrs])

where:

ivalid	An integer variable that receives the value returned by IBLNR. A –1 (true) indicates that at least one instrument specified is a valid listener. A 0 (false) indicates that no instrument specified is a valid listener.
lnrs	The optional listener specification. If no listener is specified, IBLNR uses the default listener list.

Example:

```
    LOGICAL THERE
    BYTE MESSAG (30)
10  THERE = IBLNR (15)     !Is instrument 15 on?
    IF (THERE) GO TO 20    !If so, proceed.
    PAUSE 'Turn on the equipment!'
    GO TO 10
20  CALL IBSEND ('SETUP',,15)
    CALL IBRECV (MESSAG,30,15)
    TYPE 30
30  FORMAT ('$EQUIPMENT IS NOW ON')
    END
```

Appendix A
ASCII Character Codes

ASCII Character	Octal Code	Decimal Code	Binary Code	Hexadecimal Code
NUL	000	0	0000000	00
SOH	001	1	0000001	01
STX	002	2	0000010	02
ETX	003	3	0000011	03
EOT	004	4	0000100	04
ENQ	005	5	0000101	05
ACK	006	6	0000110	06
BEL	007	7	0000111	07
BS	010	8	0001000	08
HT	011	9	0001001	09
LF	012	10	0001010	0A
VT	013	11	0001011	0B
FF	014	12	0001100	0C
CR	015	13	0001101	0D
SO	016	14	0001110	0E
SI	017	15	0001111	0F
DLE	020	16	0010000	10
DC1	021	17	0010001	11
DC2	022	18	0010010	12
DC3	023	19	0010011	13
DC4	024	20	0010100	14
NAK	025	21	0010101	15
SYN	026	22	0010110	16
ETB	027	23	0010111	17
CAN	030	24	0011000	18
EM	031	25	0011001	19
SUB	032	26	0011010	1A
ESC	033	27	0011011	1B
FS	034	28	0011100	1C
GS	035	29	0011101	1D
RS	036	30	0011110	1E
US	037	31	0011111	1F

ASCII Character	Octal Code	Decimal Code	Binary Code	Hexadecimal Code
SP	040	32	0100000	20
!	041	33	0100001	21
"	042	34	0100010	22
#	043	35	0100011	23
$	044	36	0100100	24
%	045	37	0100101	25
&	046	38	0100110	26
'	047	39	0100111	27
(	050	40	0101000	28
)	051	41	0101001	29
*	052	42	0101010	2A
+	053	43	0101011	2B
,	054	44	0101100	2C
—	055	45	0101101	2D
.	056	46	0101110	2E
/	057	47	0101111	2F
0	060	48	0110000	30
1	061	49	0110001	31
2	062	50	0110010	32
3	063	51	0110011	33
4	064	52	0110100	34
5	065	53	0110101	35
6	066	54	0110110	36
7	067	55	0110111	37
8	070	56	0111000	38
9	071	57	0111001	39
:	072	58	0111010	3A
;	073	59	0111011	3B
<	074	60	0111100	3C
=	075	61	0111101	3D
>	076	62	0111110	3E
?	077	63	0111111	3F
@	100	64	1000000	40
A	101	65	1000001	41
B	102	66	1000010	42
C	103	67	1000011	43
D	104	68	1000100	44
E	105	69	1000101	45
F	106	70	1000110	46
G	107	71	1000111	47
H	110	72	1001000	48
I	111	73	1001001	49
J	112	74	1001010	4A
K	113	75	1001011	4B
L	114	76	1001100	4C
M	115	77	1001101	4D
N	116	78	1001110	4E
O	117	79	1001111	4F
P	120	80	1010000	50
Q	121	81	1010001	51
R	122	82	1010010	52
S	123	83	1010011	53
T	124	84	1010100	54
U	125	85	1010101	55
V	126	86	1010110	56
W	127	87	1010111	57

ASCII Character	Octal Code	Decimal Code	Binary Code	Hexadecimal Code
X	130	88	1011000	58
Y	131	89	1011001	59
Z	132	90	1011010	5A
[	133	91	1011011	5B
\	134	92	1011100	5C
]	135	93	1011101	5D
^	136	94	1011110	5E
–	137	95	1011111	5F
’	140	96	1100000	60
a	141	97	1100001	61
b	142	98	1100010	62
c	143	99	1100011	63
d	144	100	1100100	64
e	145	101	1100101	65
f	146	102	1100110	66
g	147	103	1100111	67
h	150	104	1101000	68
i	151	105	1101001	69
j	152	106	1101010	6A
k	153	107	1101011	6B
l	154	108	1101100	6C
m	155	109	1101101	6D
n	156	110	1101110	6E
o	157	111	1101111	6F
p	160	112	1110000	70
q	161	113	1110001	71
r	162	114	1110010	72
s	163	115	1110011	73
t	164	116	1110100	74
u	165	117	1110101	75
v	166	118	1110110	76
w	167	119	1101111	77
x	170	120	1111000	78
y	171	121	1111001	79
z	172	122	1111010	7A
{	173	123	1111011	7B
:	174	124	1111100	7C
}	175	125	1111101	7D
˜	176	126	1111110	7E
DEL	177	127	1111111	7F

Appendix B
Command Mnemonics

B.1 IEEE Standard Command Codes

Table B–1 lists the IEEE standard command codes and shows the lines that are set on the IEEE bus when they are executed. The commands that are used to address instruments and the codes to address the instruments are explained in Tables B–2 through B–6.

Table B–1: Command Codes

Bus Signal Line(s) and Coding That Asserts the True Value of the Message

Mnemonic		Message Name	Type	Class	DIO 8	7	6	5	4	3	2	DIO 1	DAV	NRFD	NDAC	ATN	EOI	SRQ	IFC	REN
ACG	addressed command group		M	AC	Y	0	0	0	X	X	X	X	X	X	X	1	X	X	X	X
ATN	attention		U	UC	X	X	X	X	X	X	X	X	X	X	X	1	X	X	X	X
DAB	data byte	(Notes 1,9)	M	DD	D	D	D	D	D	D	D	D	X	X	X	0	X	X	X	X
					8	7	6	5	4	3	2	1								
DAC	data accepted		U	HS	X	X	X	X	X	X	X	X	X	X	0	X	X	X	X	X
DAV	data valid		U	HS	X	X	X	X	X	X	X	X	1	X	X	X	X	X	X	X
DCL	device clear		M	UC	Y	0	0	1	0	1	0	0	X	X	X	1	X	X	X	X
END	end		U	ST	X	X	X	X	X	X	X	X	X	X	X	0	1	X	X	X
EOS	end of string	(Notes 2, 9)	M	DD	E	E	E	E	E	E	E	E	X	X	X	0	X	X	X	X
					8	7	6	5	4	3	2	1								
GET	group execute trigger		M	AC	Y	0	0	0	1	0	0	0	X	X	X	1	X	X	X	X
GTL	go to local		M	AC	Y	0	0	0	0	0	0	1	X	X	X	1	X	X	X	X
IDY	identify		U	UC	X	X	X	X	X	X	X	X	X	X	X	X	1	X	X	X
IFC	interface clear		U	UC	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	X
LAG	listen address group		M	AD	Y	0	1	X	X	X	X	X	X	X	X	1	X	X	X	X
LLO	local lock out		M	UC	Y	0	0	1	0	0	0	1	X	X	X	1	X	X	X	X
MLA	my listen address	(Note 3)	M	AD	Y	0	1	L	L	L	L	L	X	X	X	1	X	X	X	X
								5	4	3	2	1								
MTA	my talk address	(Note 4)	M	AD	Y	1	0	T	T	T	T	T	X	X	X	1	X	X	X	X
								5	4	3	2	1								
MSA	my secondary address	(Note 5)	M	SE	Y	1	1	S	S	S	S	S	X	X	X	1	X	X	X	X
								5	4	3	2	1								
NUL	null byte		M	DD	0	0	0	0	0	0	0	0	X	X	X	X	X	X	X	X
OSA	other secondary address		M	SE	(OSA = SCG ∧ MSA)															
OTA	other talk address		M	AD	(OTA = TAG ∧ MTA)															
PCG	primary command group		M	—	(PCG = ACG ∨ UCG ∨ LAG ∨ TAG)															
PPC	parallel poll configure		M	AC	Y	0	0	0	0	1	0	1	X	X	X	1	X	X	X	X
PPE	parallel poll enable	(Note 6)	M	SE	Y	1	1	0	S	P	P	P	X	X	X	1	X	X	X	X
										3	2	1								
PPD	parallel poll disable	(Note 7)	M	SE	Y	1	1	1	D	D	D	D	X	X	X	1	X	X	X	X
									4	3	2	1								
PPR1	parallel poll response 1		U	ST	X	X	X	X	X	X	X	1	X	X	X	1	1	X	X	X
PPR2	parallel poll response 2		U	ST	X	X	X	X	X	X	1	X	X	X	X	1	1	X	X	X
PPR3	parallel poll response 3	(Note 10)	U	ST	X	X	X	X	X	1	X	X	X	X	X	1	1	X	X	X
PPR4	parallel poll response 4		U	ST	X	X	X	X	1	X	X	X	X	X	X	1	1	X	X	X
PPR5	parallel poll response 5		U	ST	X	X	X	1	X	X	X	X	X	X	X	1	1	X	X	X
PPR6	parallel poll response 6		U	ST	X	X	1	X	X	X	X	X	X	X	X	1	1	X	X	X
PPR7	parallel poll response 7	(Note 10)	U	ST	X	1	X	X	X	X	X	X	X	X	X	1	1	X	X	X
PPR8	parallel poll response 8		U	ST	1	X	X	X	X	X	X	X	X	X	X	1	1	X	X	X
PPU	parallel poll unconfigure		M	UC	Y	0	0	1	0	1	0	1	X	X	X	1	X	X	X	X
REN	remote enable		U	UC	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1
RFD	ready for data		U	HS	X	X	X	X	X	X	X	X	X	0	X	X	X	X	X	X
RQS	request service	(Note 9)	U	ST	X	1	X	X	X	X	X	X	X	X	X	0	X	X	X	X
SCG	secondary command group		M	SE	Y	1	1	X	X	X	X	X	X	X	X	1	X	X	X	X
SDC	selected device clear		M	AC	Y	0	0	0	0	1	0	0	X	X	X	1	X	X	X	X
SPD	serial poll disable		M	UC	Y	0	0	1	1	0	0	1	X	X	X	1	X	X	X	X
SPE	serial poll enable		M	UC	Y	0	0	1	1	0	0	0	X	X	X	1	X	X	X	X
SRQ	service request		U	ST	X	X	X	X	X	X	X	X	X	X	X	X	X	1	X	X
STB	status byte	(Notes 8, 9)	M	ST	S	X	S	S	S	S	S	S	X	X	X	0	X	X	X	X
					8		6	5	4	3	2	1								
TCT	take control		M	AC	Y	0	0	0	1	0	0	1	X	X	X	1	X	X	X	X
TAG	talk address group		M	AD	Y	1	0	X	X	X	X	X	X	X	X	1	X	X	X	X
UCG	universal command group		M	UC	Y	0	0	1	X	X	X	X	X	X	X	1	X	X	X	X
UNL	unlisten		M	AD	Y	0	1	1	1	1	1	1	X	X	X	1	X	X	X	X
UNT	untalk	(Note 11)	M	AD	Y	1	0	1	1	1	1	1	X	X	X	1	X	X	X	X

Table B–1: Command Codes (Cont.)

The 1/0 coding on ATN when sent concurrent with multiline messages has been added to this revision for interpretive convenience.

NOTES:

1. D1–D8 specify the device dependent data bits.
2. E1–E8 specify the device dependent code used to indicate the EOS message.
3. L1–L5 specify the device dependent bits of the device's listen address.
4. T1–T5 specify the device dependent bits of the device's talk address.
5. S1–S5 specify the device dependent bits of the device's secondary address.
6. S specifies the sense of the PPR.

S	Response
0	0
1	1

P1–P3 specify the PPR message to be sent when a parallel poll is executed.

P3	P2	P1	PPR Message
0	0	0	PPR1
.	.	.	.
.	.	.	.
.	.	.	.
1	1	1	PPR8

7. D1–D4 specify don't care bits that shall not be decoded by the receiving device. It is recommended that all zeroes be sent.
8. S1–S6, S8 specify the device dependent status. (DIO7 is used for the RQS message.)
9. The source of the message on the ATN line is always the C function, whereas the messages on the DIO and EOI lines are enabled by the T function.
10. The source of the messages on the ATN and EOI lines is always the C function, whereas the source of the messages on the DIO lines is always the PP function.
11. This code is provided for system use.

B.2 Mnemonic and Symbol Definitions for Table B–1

The logical state a bus signal can have is specified in the Table B–1 0, 1, Y, or X. These represent the logic states as follows:

0	=	logical zero
1	=	logical one
x	=	don't care for the coding of a received message
X	=	must not drive unless directed by another message for the coding of a transmitted message
Y	=	don't care for transmitted messages

The mnemonic used for signal type in Table B–1 are as follows:

U	=	One line message
M	=	Multiline message

The mnemonic used for class of signal in Table B–1 are:

AC	=	Addressed command
AD	=	Address (talk or listen)
DD	=	Device dependent
HS	=	Handshake signal

UC	=	Universal command
SE	=	Secondary command
ST	=	Status

Addressed Commands — Table B–2 lists the IEEE–bus command codes that affect only the instruments addressed to listen.

Table B–2: Addressed Commands

FORTRAN Routine	Octal ASCII Code	ASCII Character	Keyboard Function	Command Function	Command Mnemonic
IBGTL	001	SOH	CTRL/A	Go to local	GTL
IBSDC	004	EOT	CTRL/D	Selected device clear	SDC
IBGET	010	BS	CTRL/H	Group execute trigger	GET

Universal Commands — Table B–3 lists the IEEE–bus command codes that affect all instruments on the IEEE bus.

Table B–3: Universal Commands

FORTRAN Routine	Octal ASCII Code	ASCII Character	Keyboard Function	Command Function	Command Mnemonic
IBLLO	021	DC1	CTRL/Q	Local lockout	LLO
IBDCL	024	DC4	CTRL/T	Device clear	DCL
IBPPU	025	NAK	CTRL/U	Parallel poll unconfigure	PPU

Listener Addressing — Table B–4 contains the IEEE–bus command codes that direct instruments to listen. They are sent automatically when you specify a primary address in a listener address argument.

Table B–4: Listener Address Commands

Octal ASCII Code	ASCII Character	Keyboard Function	Command Function	Command Mnemonic
041	!	!	Listener Address 1	MLA01
042	"	"	Listener Address 2	MLA02
043	#	#	Listener Address 3	MLA03
044	$	$	Listener Address 4	MLA04
045	%	%	Listener Address 5	MLA05
046	&	&	Listener Address 6	MLA06
047	'	'	Listener Address 7	MLA07
050	(	(	Listener Address 8	MLA08
051	)	)	Listener Address 9	MLA09
052	*	*	Listener Address 10	MLA10
053	+	+	Listener Address 11	MLA11
054	,	,	Listener Address 12	MLA12
055	–	–	Listener Address 13	MLA13
056	.	.	Listener Address 14	MLA14
057	/	/	Listener Address 15	MLA15
060	0	0	Listener Address 16	MLA16
061	1	1	Listener Address 17	MLA17
062	2	2	Listener Address 18	MLA18
063	3	3	Listener Address 19	MLA19
064	4	4	Listener Address 20	MLA20
065	5	5	Listener Address 21	MLA21
066	6	6	Listener Address 22	MLA22
067	7	7	Listener Address 23	MLA23
070	8	8	Listener Address 24	MLA24
071	9	9	Listener Address 25	MLA25
072	:	:	Listener Address 26	MLA26
073	;	;	Listener Address 27	MLA27
074	<	<	Listener Address 28	MLA28
075	=	=	Listener Address 29	MLA29
076	>	>	Listener Address 30	MLA30
077	?	?	Unlisten	UNL

Talker Addressing — Table B–5 lists the IEEE–bus command codes that direct instruments to talk. They are sent automatically when you specify a primary address in a talker address argument.

Table B–5: Talker Address Commands

Octal ASCII Code	ASCII Character	Keyboard Function	Command Function	Command Mnemonic
101	A	A	Talker Address 1	MTA01
102	B	B	Talker Address 2	MTA02
103	C	C	Talker Address 3	MTA03
104	D	D	Talker Address 4	MTA04
105	E	E	Talker Address 5	MTA05
106	F	F	Talker Address 6	MTA06
107	G	G	Talker Address 7	MTA07
110	H	H	Talker Address 8	MTA08
111	I	I	Talker Address 9	MTA09
112	J	J	Talker Address 10	MTA10
113	K	K	Talker Address 11	MTA11
114	L	L	Talker Address 12	MTA12
115	M	M	Talker Address 13	MTA13
116	N	N	Talker Address 14	MTA14
117	O	O	Talker Address 15	MTA15
120	P	P	Talker Address 16	MTA16
121	Q	Q	Talker Address 17	MTA17
122	R	R	Talker Address 18	MTA18
123	S	S	Talker Address 19	MTA19
124	T	T	Talker Address 20	MTA20
125	U	U	Talker Address 21	MTA21
126	V	V	Talker Address 22	MTA22
127	W	W	Talker Address 23	MTA23
130	X	X	Talker Address 24	MTA24
131	Y	Y	Talker Address 25	MTA25
132	Z	Z	Talker Address 26	MTA26
133	[	[	Talker Address 27	MTA27
134	\	\	Talker Address 28	MTA28
135	]	]	Talker Address 29	MTA29
136	^	^	Talker Address 30	MTA30
137	–	–	Untalk	UNT

Secondary Addressing — Table B–6 lists the IEEE–bus command codes that direct alternate functions of instruments to talk or listen. When you specify a primary address followed by a secondary address in a talker argument, these command codes are sent automatically, designating which alternate function of an instrument is to talk. When you specify a primary address followed by a secondary address in a listener address argument, these command codes are sent automatically, designating which alternate function of an instrument is to listen.

Table B–6: Secondary Address Commands

Octal ASCII Code	ASCII Character	Keyboard Function	Command Function	Command Mnemonic
140	\	\	Secondary Address 0	MSA00
141	a	a	Secondary Address 1	MSA01
142	b	b	Secondary Address 2	MSA02
143	c	c	Secondary Address 3	MSA03
144	d	d	Secondary Address 4	MSA04
145	e	e	Secondary Address 5	MSA05
146	f	f	Secondary Address 6	MSA06
147	g	g	Secondary Address 7	MSA07
150	h	h	Secondary Address 8	MSA08
151	i	i	Secondary Address 9	MSA09
152	j	j	Secondary Address 10	MSA10
153	k	k	Secondary Address 11	MSA11
154	l	l	Secondary Address 12	MSA12
155	m	m	Secondary Address 13	MSA13
156	n	n	Secondary Address 14	MSA14
157	o	o	Secondary Address 15	MSA15
160	p	p	Secondary Address 16	MSA16
161	q	q	Secondary Address 17	MSA17
162	r	r	Secondary Address 18	MSA18
163	s	s	Secondary Address 19	MSA19
164	t	t	Secondary Address 20	MSA20
165	u	u	Secondary Address 21	MSA21
166	v	v	Secondary Address 22	MSA22
167	w	w	Secondary Address 23	MSA23
170	x	x	Secondary Address 24	MSA24
171	y	y	Secondary Address 25	MSA25
172	z	z	Secondary Address 26	MSA26
173	{	{	Secondary Address 27	MSA27
174	|	|	Secondary Address 28	MSA28
175	}	}	Secondary Address 29	MSA29
176	~	~	Secondary Address 30	MSA30
177	DEL	DEL		Not used

Appendix C
Building a New IBV11–A Device Handler and Modifying the IBS Library

C.1 Building a New IBV11–A Device Handler

Your Instrument Bus Subroutines software distribution kit contains four standard device handlers. The standard distributed device handlers function only with systems that meet the following two conditions:

1. The RT–11 monitor you plan to use with the device handler is either the RT–11 SJ or FB monitor distributed with the RT–11 software distribution kit, or an XM monitor generated for device timeout support and no error logging support.
2. Your system has the following IBV11–A device configuration which is called the "standard IBV11–A device configuration for the Instrument Bus Subroutines software":
    a. There is only one IBV11–A unit.
    b. The vector address of that unit is set at location 420(octal).
    c. The Control Status Register (CSR) address of that unit is set at 171420(octal) (for MINC and DECLAB–11/MNC users) or 160150(octal) (for any other PDP–11 users).

The standard device handlers are named as follows: IBMNC.SYS, IBNMNC.SYS, IBXMNC.SYS and IBXNMC.SYS.

- Use IBMNC.SYS if you have a standard MINC or DECLAB–11/MNC system and if you are using either the SJ or the FB distributed system monitors.
- Use IBNMNC.SYS if you have any other standard PDP–11 system and if you are using either the SJ or the FB distributed system monitors.
- Use IBXMNC.SYS if you have a standard MINC or DECLAB–11/MNC system and if you are using an XM (extended–memory) system monitor generated for device timeout support and no error logging support.

- Use IBXNMC.SYS if you have any other standard PDP–11 system and if you are using an XM (extended–memory) system monitor generated for device timeout support and no error logging support.

You cannot use any of the standard IBV11–A device handlers if your system fails to meet any of the above conditions. Thus, you must build a new IBV11–A device handler to meet the requirements of your specific system configuration if:

1. You plan to use the IBV11–A device handler with an RT–11 SJ or FB monitor that was created during the RT–11 system generation process, or an XM monitor generated with error logging support or without device timeout support.
2. Your system does not have one of the standard IBV11–A device configurations for the Instrument Bus Subroutines software. That is, either:
   a. Your system has multiple IBV11–A units.
   b. Your system has a single IBV11–A unit or multiple IBV11–A units, but the vector address of the first or only IBV11–A unit is not set to 420(octal), or its CSR address is not set to 171420(octal) (for MINC and DECLAB–11/MNC users) or 160150(octal) (for all other PDP–11 users).
3. You plan to use the time–out capabilities in the IB driver with an RT–11 SJ or FB monitor; device time–out support requires a system generation under these two monitors.

NOTE

For information about how to determine and set the vector and CSR addresses of an IBV11–A unit, see the *IBV11–A LSI–11/Instrument Bus Interface User's Manual.*

If you do not need to build a new IBV11–A device handler, make sure you have copied your distribution kit and applied any necessary corrections (see Chapter 10) before you attempt to use one of the distributed device handlers. Chapter 10 also contains instructions for using your distributed device handler.

If you do need to build a new device handler, you should also not attempt to do so before you have copied your distribution kit and applied any necessary corrections (see Chapter 4). After you have done so, use the procedure that follows to build a new device handler.

In this procedure, two device specifications are required, one for an "input" device, to contain the IB and IBS conditional, source, and command files, and one for an "output" device. The "output" device will probably be your active system volume, since it will contain the rebuilt IB (or IBX) device driver. However, before initiating the re–build procedure, be sure to UNLOAD and REMOVE the existing IB (or IBX) device driver from the "output" active system volume (see the *RT11 V4 System User's Guide* for a description of the UNLOAD and REMOVE commands). Next, make sure that you have at least 82 (decimal) contiguous free blocks on your "output" volume if you are building the IBS package for an SJ or an FB operating system, or at least 94 (decimal) contiguous free blocks on your "output" volume if you are building the IBS package for the XM operating system (see the *RT11 V4 System User's Guide* for a description of the DELETE and the SQUEEZE commands).

Finally, you must assign physical devices to the two required logical device names as follows.

After the monitor prompt is displayed, type:

```
.ASSIGN dvn: IN:(RET)
.ASSIGN dxm OU:(RET)
```

where:

dv	is the physical device name for the volume to contain the "input" files (e.g.: DL, DY, RK, etc.)
n	is the unit number on which the "input" volume is mounted;
dx	is the physical device name for the volume to contain the "output" files (usually this is the system device)
m	is the unit number on which the "output" volume is mounted.

This procedure requires the following files contained in the distribution kit: IBLOC.ASM, IB.MAC, and either IBDBLD.COM for RT–11 SJ and FB, or IBXBLD.COM for RT–11 XM.

The other file is called SYCND.MAC. If you are using a distributed RT–11 monitor, the file is in your RT–11 distribution kit, but it is called SYCND.DIS. This is the file you need. In Step 3 of the following procedure, you copy SYCND.DIS to your input device, and you give it the name SYCND.MAC. If you are using a version of the RT–11 monitor produced by the RT–11 system generation procedure, you created a file called SYCND.MAC during that procedure. Use SYCND.MAC when you build your new IB handler.

If you are using an RT–11 XM monitor, you will also need XM.MAC, created during the RT–11 system generation.

After you have met these requirements, rebuild your IBV11–A device handler as follows:

Step 1

Refer to Chapter 10 and edit the IBLOC.ASM file on your "input" volume to reflect the number of IBV11–A/IB11 hardware units on your system. You can have from one to eight units. Indicate the CSR (control and status register) address and vector address of your first or only IBV11–A/IB11 hardware units. More than one IBV11–A/IB11 unit must have contiguous addresses (see Chapter 3).

Step 2

Copy the 3 files named above from a copy of your Instrument Bus Subroutines software distribution volume to device "IN:".

Step 3

If you plan to use this IBV11–A device handler with the distributed RT–11 monitor, copy the file SYCND.DIS from a copy of your RT–11 distribution kit to device "IN:" giving it the name SYCND.MAC. If you plan to use this device handler with an RT–11 SJ, FB, or XM monitor created during the system generation process, copy the file SYCND.MAC which you created during that system generation process, to device "IN:". If you plan to use this device handler with an XM monitor, also copy the file XM.MAC, which is on the RT–11 distribution media, to device "IN:".

Step 4

Copy the file IBLIB.OBJ from a copy of your Instrument Bus Subroutines software distribution volume to device "ou:";

Create the new device handler by doing the following:

1. a. If the handler is for an RT–11 SJ or FB system, execute the indirect command file IBDBLD.COM. IBDBLD.COM assembles the source code for the new handler, links it to create a new IB handler called IB.SYG. After the monitor prompt appears, type

   ```
   .@IN:IBDBLD(RET)
   ```

   b. If the handler is for an RT–11 XM system, execute the indirect command file IBXBLD.COM. IBXBLD.COM assembles the source code for the new handler, links it to create a new IB handler called IBX-SYG. After the monitor prompt appears, type:

   ```
   .@IN:IBXBLD(RET)
   ```

2. Next you must rename your IB device handler file as follows:

   a. If you will be running under either the SJ or the FB operating systems, type:

   ```
   .RENAME/SYSTEM OU:IB.SYG OU:IB.SYS(RET)
   ```

   b. If you will be running under the XM (extended–memory) operating system, type:

   ```
   .RENAME/SYSTEM OU:IB.SYG OU:IBX.SYS(RET)
   ```

3. Install and load your new IB handler by typing:

```
.INSTALL IB(RET)
.LOAD IB(RET)
```

C.2 Modifying the IBS Library

If you do not have more than one IBV11–A, you do not need to modify the IBS library. Proceed to Section C.3. If you *do* have more than one IBV11–A, then you need to rebuild three files in the IBS library. The rebuild procedure requires that MACRO.SAV and LIBR.SAV (from your RT–11 distribution kit) be on your system device and that you have performed Steps 1 and 4 above.

Step 1

Copy IBUNIT.MAC, IBINIT.MAC, IBBASE.MAC, IB.ASM, and IBSBLD.COM *from* your distribution volume *to* device "IN:".

Step 2

Modify the IBS library by executing the indirect command file IBSBLD.COM. IBSBLD.COM assembles the three .MAC files named in Step 1 and updates the library. After the monitor prompt appears, type

```
.@IN:IBSBLD(RET)
```

C.3 Verifying the New Handler and Library

Referring to Section 4.4, compile, link and run the IBS installation/verification program to verify proper IBS operation after the re–build session.

Your new IBV11–A/IB11 device handler is now tailored to your system and ready for use. For more information on using this package see Section 10.3.6.

Appendix D
Sample Instrument Address Form

Instrument Name	Instrument Address	Secondary Addresses	Function

MR-S-1638-81

Appendix E
FORTRAN IB Subroutine Call Formats

Routine	Function	Format
IBARCV	Receive data asynchronously	CALL IBARCV (array, len–limit[, [tkr] [, lnrs]])
IBASND	Send data messages asynchronously	CALL IBASND (msg–array[, [msg–len] [, lnrs]])
IBAXFR	Transfer data asynchronously	CALL IBAXFR ([len–limit][, [tkr] [, lnrs]])
IBCMD	Programmed command	CALL IBCMD (n [, lnrs])
IBDCL	Device clear	CALL IBDCL
IBDEV	Provides list of instrument bus addresses that are capable of generating service requests	CALL IBDEV (dev–lst)
IBERRF	Read error flag	maperr = IBERRF ([n1 [[, ..., n16]])
IBFREE	Free transfer of data	CALL IBFREE ([tkr] [, lnrs])
IBGET	Group execute trigger	CALL IBGET ([lnrs])
IBGTL	Go to local	CALL IBGTL ([lnrs])
IBIFC	Interface clear	CALL IBIFC
IBLLO	Local lockout	CALL IBLLO
IBLNR	Check for valid listeners	ivalid = IBLNR ([lnrs])
IBPPD	Parallel poll disable	CALL IBPPD ([lnrs])
IBPPE	Parallel poll enable	CALL IBPPE (isense, line[, lnrs])
IBPPL	Parallel poll	ireslt = IBPPL ([n1 [, ..., n8]])
IBPPU	Parallel poll unconfigure	CALL IBPPU
IBRCVS	Obtain status of a data transmission initiated by IBRECV, IBARCV, IBXFER, or IBAXFR	Call IBRCVS ([iend]) or iend = IBRCVS ( )

Routine	Function	Format
IBRDA	Remote disable	CALL IBRDA
IBRECV	Receive data	CALL IBRECV (array, len–limit[, [tkr] [, lnrs]]) len = IBRECV (array, len–limit[, [tkr] [, lnrs]])
IBREMO	Remote status	nold = IBREMO (n)
IBREN	Remote enable	CALL IBREN
IBSDC	Selected device clear	CALL IBSDC ([lnrs])
IBSEND	Send data messages	CALL IBSEND (msg–array[, [msg–len] [, lnrs]])
IBSEOI	Send data messages with end or identify	CALL IBSEOI (msg–arry[, [msg–len] [, lnrs]])
IBSPL	Serial poll	index = IBSPL ([istat] [, tkr [, tkr [, ...]]])
IBSRQ	Name service routine	CALL IBSRQ ([subroutine–name],[ipaddr],[isadd], [istat],[idelay],[dev–lst])
IBSRQF	Test service request flag	CALL IBSRQF ([iflag]) iflag = IBSRQF ( )
IBSTER	Set error handling characteristics	CALL IBSTER (nerr, ncount)
IBSTS	Check instrument status	istat = IBSTS ([tkr])
IBTERM	Define terminating characters	CALL IBTERM ([chr–val[, chr–val, ...]])
IBTIMO	Set the timeout	CALL IBTIMO ([nticks]) nold = IBTIMO ([nticks])
IBUNIT	Select IBV11–A/IB11 unit	CALL IBUNIT ([nunit]) nold = IBUNIT ([nunit])
IBUNL	Remove listeners from IEEE bus	CALL IBUNL
IBUNT	Remove talkers from IEEE bus	CALL IBUNT
IBWAIT	Wait for data transmission to complete	CALL IBWAIT nbytes = IBWAIT ()
IBXFER	Transfer data	CALL IBXFER ([len–limit] [, [tkr] [, lnrs]]) len = IBXFER ([len–limit][, [tkr] [, lnrs]])

Appendix F
Unsigned Integers

Certain IB routine arguments benefit from the extended positive range available to unsigned integers. Since Fortran uses only signed integers you need to understand how to convert from one to the other in order to fully use the range available to these arguments.

Unsigned integer arguments cover the range from 0 to 65535, yet FORTRAN single precision positive integers only cover the range from 0 to 32767. This apparent conflict is a matter of interpretation of the 16 bit quantity.

A 16 bit binary number can represent decimal integers from 0 to 65535. In Fortran this range is divided into two parts: values from 0 to 32767 are interpreted as positive integers: values in the rest of the range are interpreted as negative integers. The Fortran negative number range from -32768 to -1 corresponds directly to the unsigned number range from 32768 to 65535.

For arguments in the range 0 to 32767 no conversion is necessary. For larger numbers up to 65535 use the following equation where R is the unsigned number and N is the signed integer to use in the subroutine call. Note that FORTRAN cannot handle integers larger than 32767 so R must be a floating point variable when used in a program.

```
N = R - 65536.
```

The following example uses two Fortran statements to convert the unsigned integer (represented as a real number) in R to a signed integer in N.

```
C     If R ≤ 32767, no conversion is
      necessary
      IF (R .LE. 32767.) N = R

C     If R > 32767, convert to negative number
      IF (R .GT. 32767.) N = R - 65536.
```

Index

Instrument Bus Subroutines
Programmer's Reference Manual
AD–5613C–T1

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) ____________

Name ____________ Date ____________

Organization ____________ Telephone ____________

Street ____________

City ____________ State ____________ Zip Code or Country ____________

------ Do Not Tear – Fold Here and Tape --

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE PUBLICATIONS
200 FOREST STREET MR1–2/L12
MARLBOROUGH, MASSACHUSETTS 01752

----- Do Not Tear – Fold Here and Tape --

UPDATE NOTICE

Instrument Bus Subroutines Programmer's Reference Manual

AD–5613C–T1

August 1982

Insert this Update Notice in the *Instrument Bus Subroutines Programmer's Reference Manual* to maintain an up-to-date record of changes to the manual.

Changed Information

The changed pages contained in this update package reflect the change from Version 2.0 to 2.1 of the IBS software and corrections to the documentation.

The instructions for inserting this update start on the next page.

digital software

INSTRUCTIONS AD–5613C–T1

The following list of page numbers specifies which pages are to be placed in the *Instrument Bus Subroutine Programmer's Reference Manual* as replacements for, or additions to, current pages.

Title page Copyright page	4–9 4–10	10–1 10–2
ix x	5–9 5–10	B–3 B–4
2–3 2–4	7–3 7–4	C–1 C–5
2–7 2–8	8–1 8–6	E–1 E–2
4–1 4–2	9–5 9–6	Index-3 Index-6

KEEP THIS UPDATE NOTICE IN YOUR MANUAL TO MAINTAIN AN UP-TO-DATE RECORD OF CHANGES.

TYPE AND IDENTIFICATION OF DOCUMENTATION CHANGES.
Five types of changes are used to update documents contained in the software manuals. Change symbols and notations are used to specify where, when, and why alterations were made to each update page. The five types of update changes and the manner in which each is identified are described in the following table.

The Following Symbols and/or Notations	Identify the Following Types of Update Changes
1. Change bar in outside margin; version number and change date printed at bottom of page.	1. Changes were required by a new version of the software being described.
2. Change bar in outside margin; change date printed at bottom of page.	2. Changes were required to either clarify or correct the existing material.
3. Change date printed at bottom of page.	3. Changes were made for editorial purposes but use of the software is not affected.
4. Bullet (●) in outside margin; version number and change date printed at bottom of page.	4. Data was deleted to comply with a new version of the software being described.
5. Bullet (●) in outside margin; change date printed at bottom of page.	5. Data was deleted to either clarify or correct the existing material.

August 1982