MINC-11

Book 5:
MINC IEEE Bus Programming

November 1978

This manual describes the MINC routines that control the IEEE-488-1975 general purpose instrument bus.

Order Number AA-D801A-TC

MINC-11

VERSION 1.0

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

First Printing, November 1978

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

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10
MINC-11	DECSYSTEM-2020	

CONTENTS

FIGURES

PREFACE

This manual describes the MINC routines that control the IEEE bus, the general purpose instrument bus described in standard 488-1975 of the Institute of Electrical and Electronic Engineers, *IEEE Standard Digital Interface for Programmable Instrumentation* (also known as standard MC 1.1-1975 of the American National Standards Institute). Programs use these routines to control instruments on the bus and to send data to and receive data from these instruments.

This manual has two major parts. Part 1 describes the IEEE bus and what the IEEE routines do. Part 2 describes how to use the IEEE routines: these individual routine descriptions are arranged alphabetically for easy reference. Part 1 has two levels of discussion: Chapter 1 is an introduction to the bus, but contains enough information for many applications, while Chapter 2 is more detailed and advanced.

PART 1

INTRODUCTION TO IEEE BUS PROGRAMMING

CHAPTER 1
FUNDAMENTAL CONCEPTS

WHAT IS THE IEEE BUS?

The *IEEE bus* permits a computer to communicate with instruments through strings of characters, called *messages*. The computer can send an instrument a string that tells the instrument what to do. The instrument can send back a string of information about data it has collected.

Since this bus became an IEEE standard and an ANSI standard in 1975, hundreds of instruments with diverse applications have been built to be connected to it. Any instrument that conforms to the IEEE standard can communicate with MINC through this bus, though an instrument can conform to the standard without having all of the capabilities defined in the standard.

Physically, the IEEE bus is a cable of 16 wires, or *bus lines*, which are shared by MINC and all instruments on the bus. The cable, connectors to the cable, and electrical requirements are defined by the standard and are described in Chapter 6 of Book 7.

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 characters sent on the bus. Three more lines, called the *handshake lines*, are used to be sure 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.

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 may *listen* to the message being sent by the talker.

MINC 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 MINC. MINC controls all bus activity, and it must be the only controller of the bus. This means that no other calculator or computer, not even another MINC, can be a controller on this IEEE bus.

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

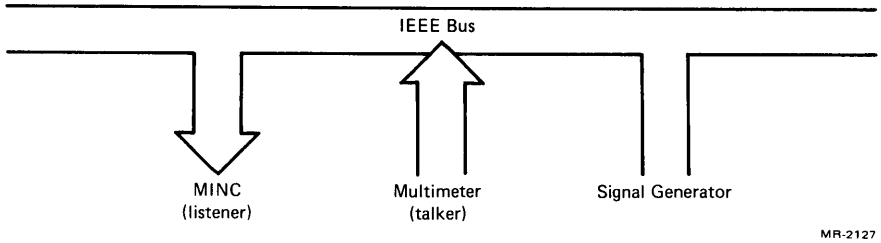

Figure 1. MINC Receives a Message

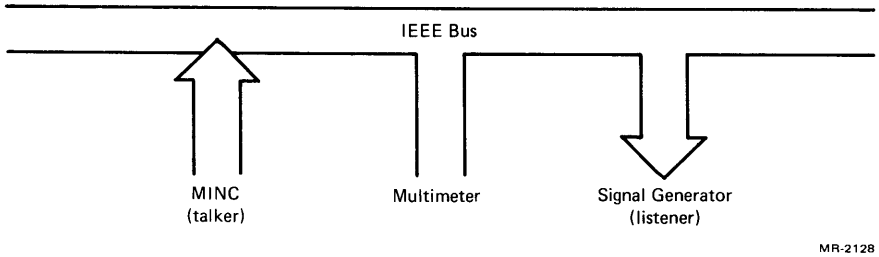

Figure 2. MINC Sends a Message

IEEE Bus Addresses

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

MR-2126

Figure 3. Sample Address Record

Instrument Addresses

Each instrument on the bus has a number between 0 and 30 that MINC uses to identify the instrument when telling it to talk or listen. This number is the instrument's *address* and can be set with switches located on the instrument itself. Chapter 6 of Book 7 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 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 MINC 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 3, an amperage 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 3. Be sure that no two instruments have the same address. Blank forms are provided at the end of this chapter and at the end of the book.

The Interface

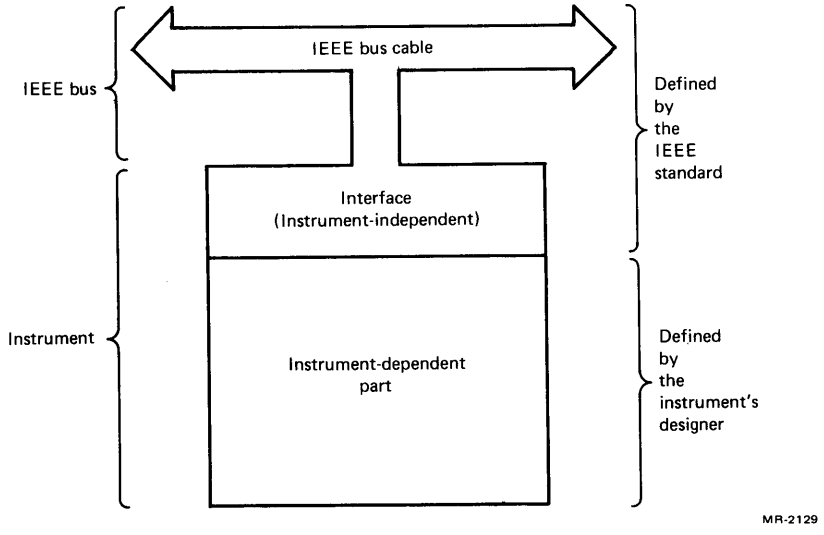

Figure 4. An Instrument's Interface

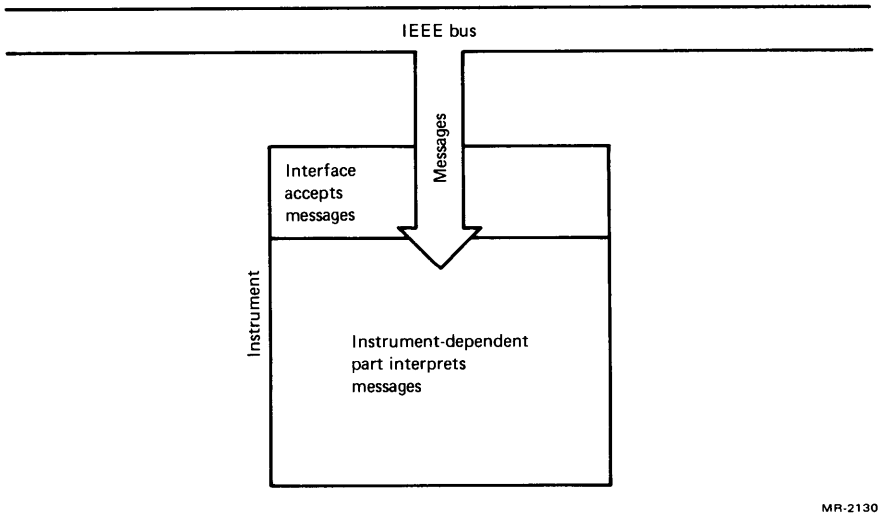

Figure 5. An Instrument Listens

Part of each instrument on the IEEE bus is defined by the IEEE standard and is thus instrument-independent. This part is called the instrument's *interface* to the bus. The rest of the instrument is instrument-dependent. It is defined not by the standard but by the instrument's designer. These parts are illustrated in Figure 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 MINC tells the instrument to listen, the instrument's

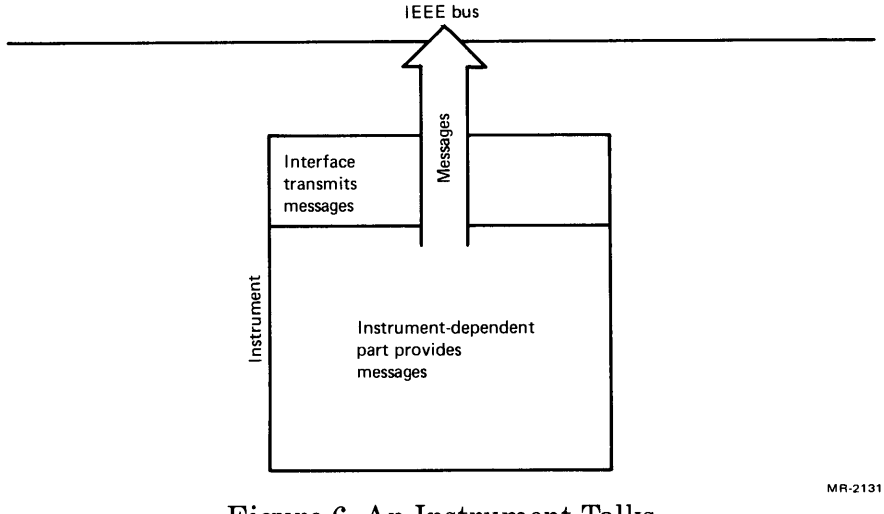

Figure 6. An Instrument Talks

interface passes any subsequent messages sent on the bus to the instrument-dependent part of the instrument. This is illustrated in Figure 5.

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

MINC controls the bus by sending *commands*, which are instructions to the interfaces on the bus. This is illustrated in Figure 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 MINC, the bus controller, can send commands. MINC uses commands to tell instruments to talk or listen.

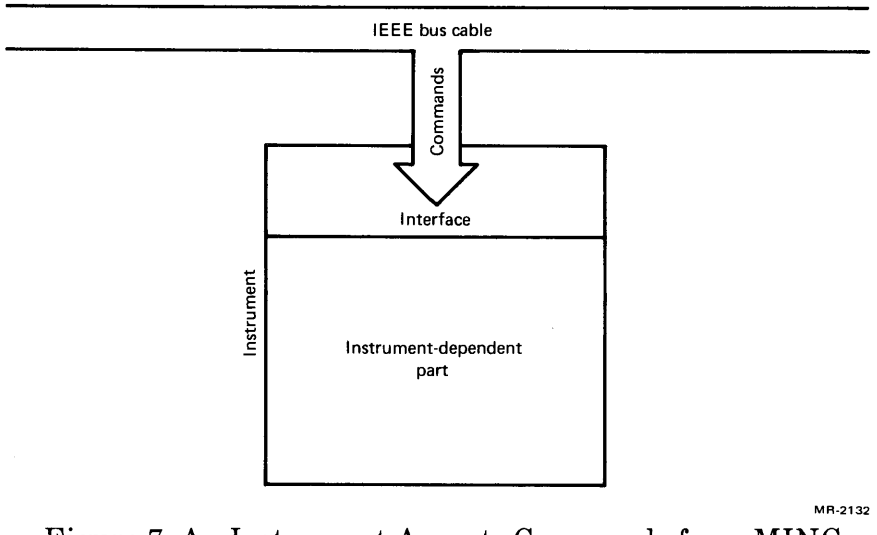

Figure 7. An Instrument Accepts Commands from MINC

MESSAGE ROUTINES

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

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."

Sending Messages

The SEND routine sends a message to one or more instruments on the IEEE bus. MINC 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 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.

For example, a statement that sends the message "V43.8F1250" to the signal generator above is:

```
SEND("V43.8F1250",2)
```

where 2 is the signal generator's address.

One of the five general management bus lines is known as the END line. This line is reserved by the standard so that the talker can 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 line is set. The SEND routine sets this bus line while sending the last character of the string specified by your program. In the example above, it sets the END line while sending the character "0."

Receiving Messages

The RECEIVE routine receives a message from an instrument on the IEEE bus. MINC 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:

```
RECEIVE(V$,,1,203)
```

where 1 is the multimeter's address, 203 indicates secondary address 3 (a voltage reading), and the message string (in this case "V+4.382E+01") is stored in the variable V$.

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

1. The talker sets the END bus line while sending a character. For example, the multimeter above might set the END line while it sends the character "1."

2. The talker sends a *terminator*, a character MINC does not store as part of a message but recognizes only as an end-of-message indicator. Normally, MINC recognizes a carriage return or a line feed as a terminator, but you can change which characters are terminators by using the SET_TERMINATORS routine. For example, the multimeter might follow the "1" character with a carriage return character. Note that if it did this, it would not set the END line while sending the "1." Depending on how it was designed, the instrument might or might not set the END line while sending the carriage return character.

3. The number of characters sent in this message reaches a

limit set by your program in the RECEIVE statement. For example, if the multimeter is not designed to set the END line or send a terminator, your program should specify eleven (the number of characters in the string "V+4.382E+01") as the maximum number of characters in the message.

Transferring Messages

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

The TRANSFER routine should only be used with compatible instruments; the vocabulary of the listener you specify should be able to interpret the message sent by the talker you specify.

TRIGGERING

The TRIGGER_INSTR 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 RECEIVE routine.

TRIGGER_INSTR 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 SEND 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 simultaneously and can then obtain the respective readings one at a time by using RECEIVE statements.

STATUS

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

As controller of the IEEE bus, MINC can ask instruments for their status by conducting either a *serial poll* or a *parallel poll*. In a serial poll, instruments one at a time report status information 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, instruments 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 is able to 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 MINC without having to wait to be polled. It can tell MINC 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 MINC.

Since bus lines are shared by all instruments, MINC 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. 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.

Serial Polls

The SERIAL_POLL 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 MINC 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 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 MINC. 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.

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

Figure 8. Serial Poll Response

Service Requests

Your program can detect a service request in two ways: by periodically using the TEST_SRQ routine to test whether or not the SRQ line is set or by using SRQ_SUBROUTINE to designate a service subroutine. Whenever any instrument sets the SRQ line, this subroutine is automatically invoked as soon as the current statement has finished executing.

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 GOSUB 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 GOSUB 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 "Service Subroutines" and "Program Dynamics for Control Programs" sections of Book 6.

Having detected a service request, your program still does not know which instrument is requesting service. Part of the information reported by an instrument during a serial poll, however, is whether or not that instrument is currently setting the SRQ line. To determine the source of a service request, therefore, your program should use the SERIAL_POLL routine, which conducts a serial poll until it polls an instrument that reports that it is requesting service.

When MINC serially polls an instrument that is requesting service, the instrument stops setting the SRQ line. Therefore, when your program detects a service request, it should conduct a

serial poll even if it knows which instrument is requesting service, so that the instrument will free the SRQ line for future use. If this is not done, MINC cannot detect any new service requests because the SRQ line is always set.

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 RECEIVE routine. Another instrument might request service when it is out of paper; your program would type a warning on the terminal.

Parallel Polls

The PAR_POLL 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 response enabled either by local controls or by MINC, 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 ENABLE_PAR_POLL routine enables the parallel poll response of instruments whose response can be enabled by the bus controller (MINC).

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 ENABLE_PAR_POLL routine, however, it can be disabled by the DISABLE_PAR_POLL routine or by the DISABLE_ALL_PAR_POLL routine. The DISABLE_PAR_POLL routine affects selected instruments, while the DISABLE_ALL_PAR_POLL 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 9.

Data Line	Value
7	0
6	0
5	0 if instrument 17's status bit is 1 1 if instrument 17's status bit is 0
4	0
3	0
2	0 if instrument 15's status bit is 0 1 if instrument 15's status bit is 1
1	0
0	0

MR-2124

Figure 9. Example of a Parallel Poll Response

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.

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 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 MINC, this line is clear, so all instruments on the bus are in the local state. The first IEEE bus routine to execute after MINC is started sets the REN

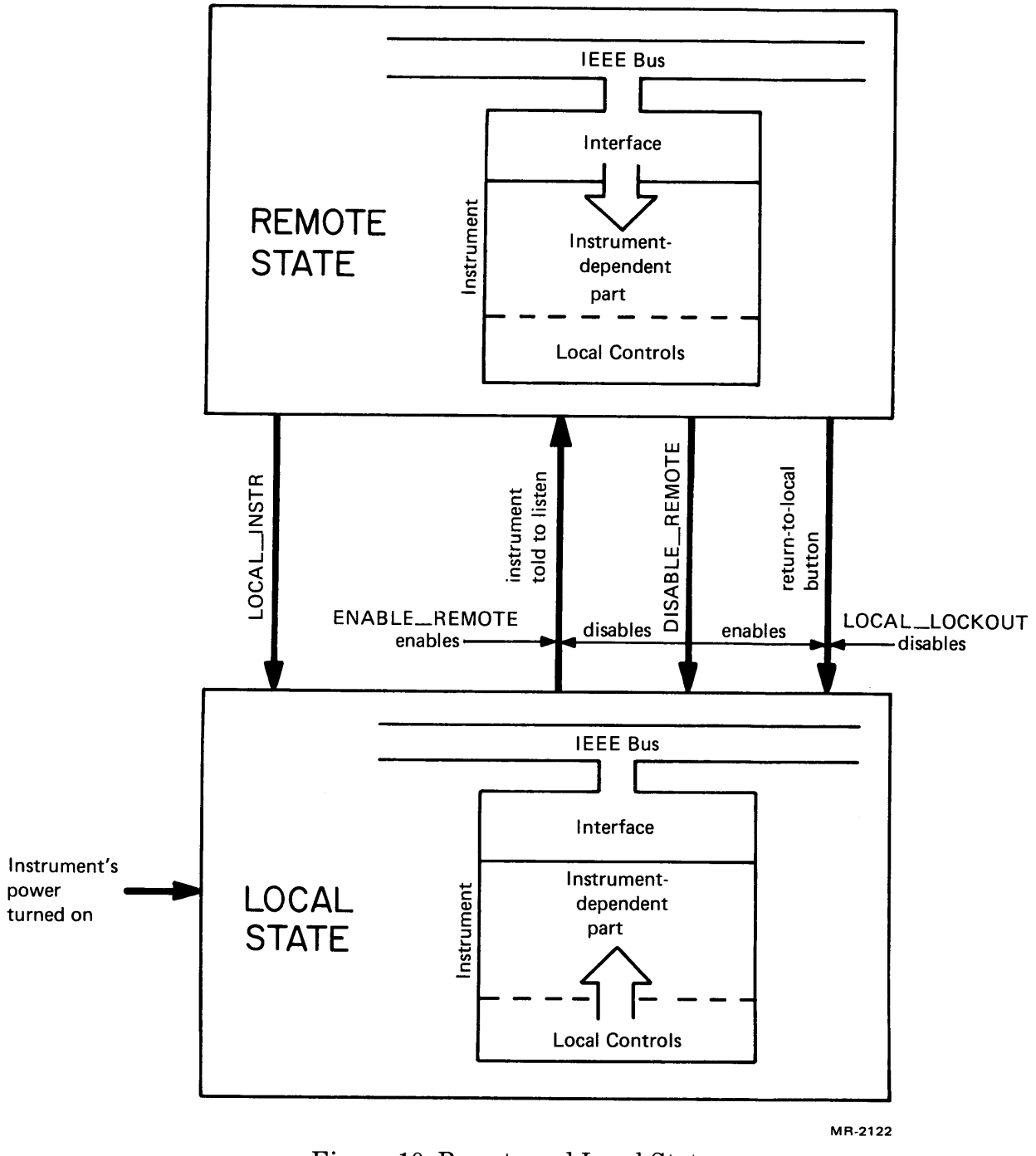

Figure 10. Remote and Local States

line automatically. This line remains set unless your program turns it off. An instrument enters the remote state when MINC tells it to listen while the REN bus line is set.

The DISABLE_REMOTE routine clears the REN bus line, causing all instruments on the bus to enter the local state. The LOCAL_INSTR routine causes selected instruments to enter the local state, but it does not change the REN line. The ENABLE_REMOTE routine sets the REN line, and the TEST_REMOTE 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 LOCAL_LOCKOUT 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. LOCAL_INSTR and DISABLE_REMOTE still cause instruments to enter the local state even if LOCAL_LOCKOUT has been executed. DISABLE_REMOTE cancels the effect of LOCAL_LOCKOUT, 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.

RESETS

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

Clearing Interfaces

The IEEE_BUS_CLEAR 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. 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 (see "Remote and Local," page 13). If LOCAL_LOCKOUT 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 (see "Remote and Local," page 13). Because the REN bus line is set, however, 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 a SEND routine) is not affected by this routine.

The first IEEE bus routine to execute after MINC is started calls IEEE_BUS_CLEAR automatically. You can use IEEE_BUS_CLEAR at the beginning of your program to undo any effect that previous IEEE bus routines have had on the interfaces.

Clearing Instruments

Two routines clear the instrument-dependent parts of instruments on bus. The ALL_INSTR_CLEAR routine clears all bus instruments, and the INSTR_CLEAR routine clears only selected instruments. These routines do not clear the instruments' 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. (CAUTION: Never turn on an instrument's power while MINC is running!)

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

IEEE Bus Addresses

Instrument	Address	Secondary Addresses

MR-2143

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 many 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: 8 data lines, 3 handshake lines, and 5 general bus management lines. These are illustrated in Figure 11. Each line is set when it is grounded, so that a bus line is set if one or more instruments sets it. A line is clear only if no instrument sets it.

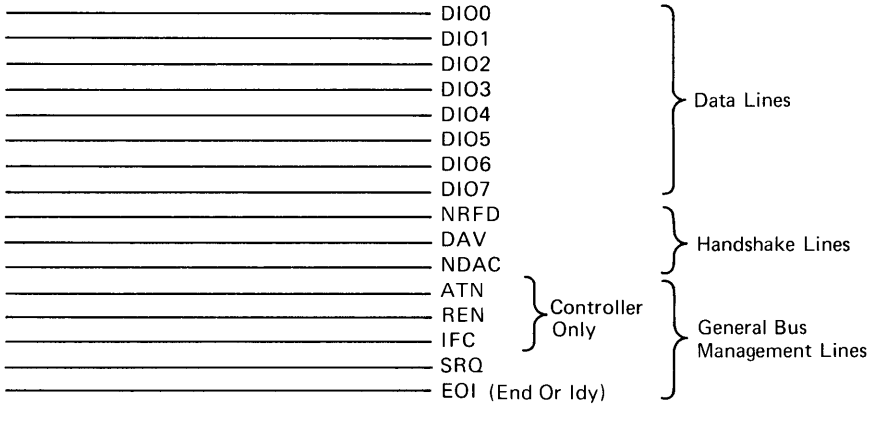

Figure 11. The IEEE Bus Lines

COMMANDS: THE ATN LINE

MINC controls the bus by sending *commands* to the interfaces. A command is a character MINC sends on the bus while the attention (ATN) bus line is set. Only MINC 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, 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.

Instrument Addressing

Some commands tell a certain instrument to talk or listen. When your program specifies a talker address in an IEEE routine, MINC 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 MINC 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 UNTALK command; routines in which MINC is the talker send the UNTALK command so that no other instrument talks when MINC clears the ATN line.

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

In a similar way, when your program specifies a listener address in an IEEE routine, MINC 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 MINC 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 MINC sends the UNLISTEN command, after which no instrument is a listener.

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

Universal and Addressed Commands

The other commands MINC sends are classified as either universal commands or addressed commands. *Universal commands* affect all instruments, while *addressed commands* affect only listeners. The names of these commands are listed in Figure 14 (page 24): the commands are discussed in Part 2 in the "Operation" section of the routines that use them.

Control Conflict

Only MINC, the bus controller, can set or clear the ATN, IFC, and REN bus lines. If another controller changes any of these three lines, MINC produces a "?MINC-F-Conflict over control of the bus" error when the next IEEE routine is invoked. MINC also produces this error if the total cable length of your IEEE bus is too long (see chapter 6 of Book 7).

THE 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 line to mark the end 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.

Parallel Polls: IDY

MINC 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.

Fragmented Messages END

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

Because MINC allows no more than 255 characters in a string, one SEND statement cannot send a message longer than 255 characters. The message must be sent in fragments, with each message fragment containing up to 255 characters. The END 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 SEND_FRAGMENT for every message fragment except the last, and use SEND for the last fragment of the message.

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 MINC. The data lines can represent numbers in the range 0 to 255. Each data line is associated with a value, as shown in Figure 12. 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).

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

MR-2125

Figure 12. Values Associated with the Data Lines

Message Codes

Each character is associated with a unique number in the range 0 to 127, called its ASCII value. A talker sends a message character by setting the data lines so that they represent the ASCII value of that character. Figure 13 lists characters and their representation on the data lines. For example, the character "A" is represented by data lines 0 and 6 set. Book 2 discusses ASCII values in detail.

To send a nonprinting character as part of a message, use the CHR$ function described in Book 3 with the character's ASCII value and concatenate that string with the printing characters in the message. For example, the string "ABC" followed by a carriage return character is "ABC"+CHR$(13), because 13 is the ASCII value of the carriage return character.

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

Command Codes

Each command has a numeric code and therefore a representation on the data lines. These representations are shown in Figure 14. This figure shows all of the command codes used by the

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

Figure 13. ASCII Character Codes

IEEE routines. Figure 15 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.

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 interpret-

	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

Figure 14. Command Codes

ed as parallel poll enable commands if they follow a parallel poll configure command (one of the addressed commands).

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
0	0	1	Instrument Address					MLA
0	1	0	Instrument Address					MTA
0	1	1	Secondary Address					MSA
0	1	1	0 (Condition)	*	*	*	* (Data Line)	PPE

* = 0 or 1

MR-2134

Figure 15. Command Code Format

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 16. Use the TEST_BIT routine to test a particular bit in the numeric result of the poll and thus determine whether the corresponding data line was clear or set. TEST_BIT is a lab module routine and is described in Book 6.

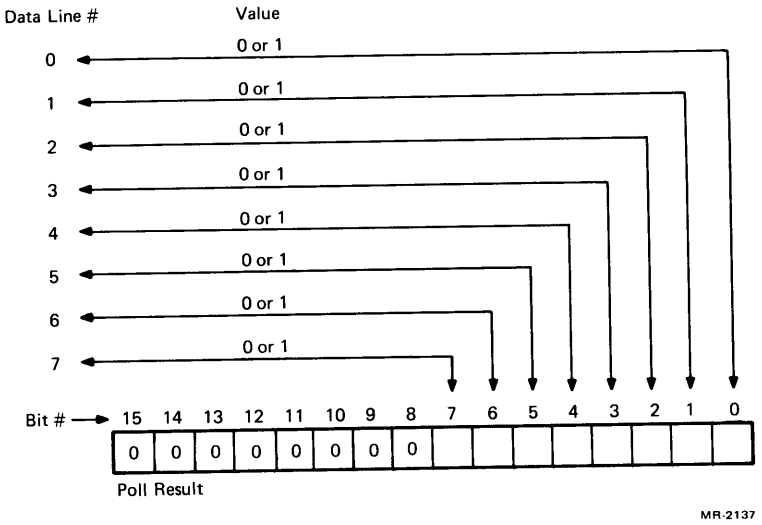

Figure 16. Poll Result Bits Correspond to Data Lines

THE HANDSHAKE LINES

The Handshake

Three bus lines provide a *handshake* mechanism to assure 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 MINC 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.

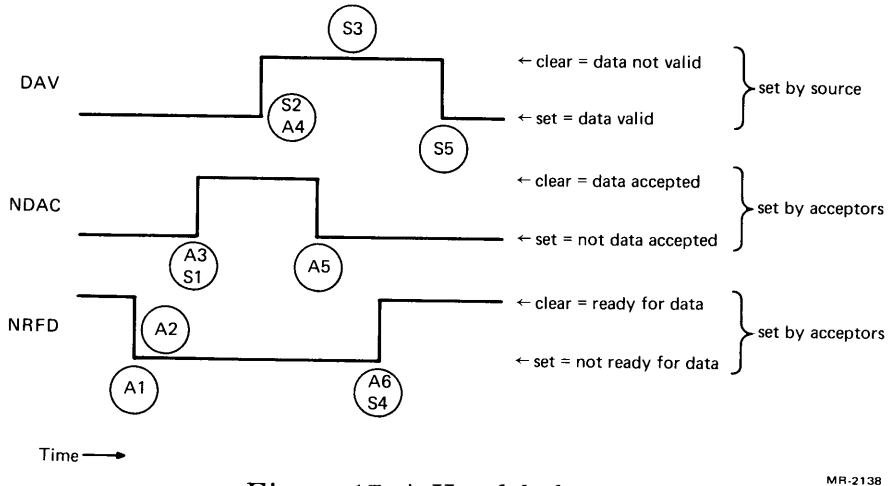

Figure 17. A Handshake

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 17 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 becomes set 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 in preparation for the next handshake. This line becomes set when the first acceptor sets it.

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

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 MINC to

produce a "?MINC-F-Listener is not on the bus" error. This error occurs in an IEEE routine under the following circumstances:

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 TEST_LISTENERS 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 detects that the acceptor handshake is missing when it tries to send the message. 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 (SERIAL_POLL routine only).

The INSTR_TIME_LIMIT routine allows you to set a time limit for each handshake, so tht MINC produces a "?MINC-F-Instrument time limit exceeded" error instead of waiting indefinitely for the next character from the talker when one of the above conditions occurs. The time limit is 120 system clock ticks (at 50 or 60 ticks per second, depending on the frequency of your electrical power) unless you change it with the INSTR_TIME_LIMIT routine.

The Idle State

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

MINC COMMANDS

Certain MINC commands issue an UNLISTEN command if any instrument has been told to listen and an UNTALK command if any instrument has been told to talk. These commands then clear the NRFD line and the REN line and then set the REN line again if it was already set. This causes all instruments to enter the local state and makes all return-to-local buttons operative. These MINC commands are discussed in Book 3 and are listed below:

BYE
COLLECT
COPY
CREATE
DATE
DIRECTORY
DUPLICATE
EDIT
EXTRA_SPACE
HELP
INITIALIZE
INSPECT
NORMAL_SPACE
RESTART
TIME
TYPE
VERIFY

PART 2
ROUTINES

SYNTAX CONVENTIONS

The routine descriptions in Part 2 use the structural framework described in the "Syntax Conventions" section of Book 6 except for the differences listed below. As in Book 6, the required parts of the statement form are printed in black and the optional parts are printed in blue.

1. All IEEE routines use whole number values. A real number can always be specified in any numeric argument, of course, but IEEE routines use the value of the next lowest whole number, disregarding any fractional part of the number.

2. The "Operation" section of the IEEE routine descriptions usually consists of two paragraphs. The first paragraph describes briefly the capability of the routine and refers to relevant sections in Chapter 1. The second paragraph describes in detail how the routine uses the bus and refers to relevant sections in Chapter 2.

3. IEEE routine descriptions have no "Configuration" section.

4. Some IEEE routines have plural argument descriptors, which represent a list of arguments, separated by commas, rather than a single argument.

5. In the "talker" and "listener" arguments, secondary addresses 0 through 30 are specified by the numbers 200 through 230 in order to distinguish them from primary addresses 0 through 30, which are specified by the numbers 0 through 30.

ALL_INSTR_CLEAR

Clear All Bus Instruments

Operation

The ALL_INSTR_CLEAR routine clears the instrument-dependent part of every instrument on the bus, returning each to a clear state defined by its manufacturer. (See "Clearing Instruments," page 16.)

This routine sends the "device clear" universal command. (See "Commands: the ATN Line," page 20, and "Command Codes," page 22.)

Statement Form

ALL_INSTR_CLEAR

Argument	*Type of Argument*	*Valid Values*	*Default Value*
	This routine has no arguments.		

Example ALL_INSTR_CLEAR
Result Clear all instruments on the bus.

Argument Descriptions

This routine has no arguments.

Related Routines

IEEE_BUS_CLEAR ALL_INSTR_CLEAR does not affect the interfaces of instruments. IEEE_BUS_CLEAR is a complementary routine that clears the interfaces of all instruments on the bus, but does not affect their instrument-dependent parts.

INSTR_CLEAR The INSTR_CLEAR routine clears the instrument-dependent parts of instruments just as ALL_INSTR_CLEAR does, but INSTR_CLEAR affects only the instruments you specify instead of all bus instruments.

Restrictions

You can use the ALL_INSTR_CLEAR routine at the beginning of your program to undo effects that the message routines of previous programs have had on bus instruments.

Each instrument must be capable of being cleared by the bus controller. If an instrument's manufacturer has not implemented this part of the standard, then trying to clear the instrument with ALL_INSTR_CLEAR does not produce an error but also does not clear that instrument; other bus instruments are still cleared.

Errors

?MINC-F-Invalid argument

?MINC-F-Listener is not on the bus

No instrument is on the bus.

Example

See the example for the SEND routine.

DISABLE_ALL_PAR_POLL

Disable Parallel Poll Response of All Instruments

Operation

The DISABLE_ALL_PAR_POLL routine disables the parallel poll response of every instrument whose response has been enabled by the ENABLE_PAR_POLL routine. These instruments no longer respond to parallel polls. (See "Parallel Polls," page 12.)

This routine sends the "parallel poll unconfigure" universal command. (See "Commands: the ATN Line," page 20, and "Command Codes," page 22.)

Statement Form

DISABLE_ALL_PAR_POLL

Argument	*Type of Argument*	*Valid Values*	*Default Value*
	This routine has no arguments.		

Example DISABLE_ALL_PAR_POLL
Result All instruments ignore subsequent parallel polls.

Argument Descriptions

This routine has no arguments.

Related Routines

DISABLE_PAR_POLL The DISABLE_PAR_POLL routine has the same effect as the DISABLE_ALL_PAR_POLL routine, except that DISABLE_PAR_POLL affects only the instruments you specify.

ENABLE_PAR_POLL The ENABLE_PAR_POLL routine enables selected instruments to respond to parallel polls.

PAR_POLL The PAR_POLL routine conducts a parallel poll. Instruments whose parallel poll response has been disabled by DISABLE_ALL_PAR_POLL or by DISABLE_PAR_POLL do not respond to the poll.

Restrictions

The parallel poll response of some instruments can be enabled and disabled only by local controls. Trying to disable the parallel poll response of such an instrument with DISABLE_ALL_PAR_POLL does not cause an error, but also does not disable its parallel poll response if the response has been enabled by local controls. The parallel poll responses of other instruments, however, are still disabled.

DISABLE_ALL_PAR_POLL

Errors

?MINC-F-Invalid argument

?MINC-F-Listener is not on the bus

No instrument is on the bus.

Example

See the example for the ENABLE_PAR_POLL routine.

DISABLE_PAR_POLL

Disable Parallel Poll Response of Selected Instruments

Operation

The DISABLE_PAR_POLL routine disables the parallel poll response of every specified instrument whose response has been enabled by the ENABLE_PAR_POLL routine. These instruments no longer respond to parallel polls. (See "Parallel Polls," page 12.)

This routine tells the specified instruments to listen by sending the "unlisten" command and the appropriate "my listen address" and "my secondary address" commands. It then sends the "parallel poll configure" addressed command (which tells listeners to interpret commands of the form 011***** as parallel poll commands instead of as "my secondary address" commands), the "parallel poll disable" command, and the "untalk" command (which tells instruments to stop interpreting commands of the form 011***** as parallel poll commands). (See "Commands: the ATN Line," page 20, and "Command Codes," page 22.)

Statement Form

DISABLE_PAR_POLL(listeners)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
listeners	numeric expressions	0 to 30; 200 to 230	required argument

Example DISABLE_PAR_POLL(3)
Result Instrument 3 no longer responds to parallel polls.

Example DISABLE_PAR_POLL(4,7)
Result Instruments 4 and 7 no longer respond to parallel polls.

Argument Descriptions

listeners The addresses of the instruments whose parallel poll response is to be disabled.

Values: 0 to 30; 200 to 230

Default: required argument

The listeners argument is a list of instrument addresses (0 to 30). Each address can be followed by a secondary address (200 to 230), though secondary addresses are not usually meaningful in this routine. Separate addresses with commas.

Related Routines

DISABLE_ALL_PAR_POLL The DISABLE_ALL_PAR_POLL routine has the same effect as the DISABLE_PAR_POLL routine, except that DISABLE_ALL_PAR_POLL affects all bus instruments.

ENABLE_PAR_POLL The ENABLE_PAR_POLL routine enables selected instruments to respond to parallel polls.

PAR_POLL The PAR_POLL routine conducts a parallel poll. Instruments whose parallel poll response has been disabled by DISABLE_PAR_POLL or DISABLE_ALL_PAR_POLL do not respond to the poll.

Restrictions

The parallel poll response of some instruments can be enabled and disabled only by local controls. Trying to disable the parallel poll response of such an instrument with DISABLE_PAR_POLL does not cause an error, but also does not disable its parallel poll response if the response has been enabled by local controls. The parallel poll responses of any other instruments specified, however, are still disabled.

Errors

?MINC-F-Invalid instrument address

?MINC-F-Listener is not on the bus

The instrument you specified as a listener is either not on the bus, turned off, not designed to listen, set locally to "talk only," or not at the address specified.

Example

See the example for the ENABLE_PAR_POLL routine.

DISABLE_REMOTE

Put All Bus Instruments in the Local State

Operation

DISABLE_REMOTE clears the "remote enable" (REN) bus line. All instruments go to the local state, in which they use input from their local controls instead of from the IEEE bus. (See "Remote and Local," page 13.)

Statement Form

DISABLE_REMOTE

Argument	*Type of Argument*	*Valid Values*	*Default Value*
	This routine has no arguments.		

Example DISABLE_REMOTE
Result Clears the REN bus line. All instruments enter the local state.

Argument Descriptions

This routine has no arguments.

Related Routines

ENABLE_REMOTE ENABLE_REMOTE is a complementary routine to DISABLE_REMOTE. It sets the REN bus line so that instruments enter the remote state as they are told to listen. Because IEEE message routines have no effect on instruments when the REN line is clear, the next IEEE routine after DISABLE_REMOTE is usually ENABLE_REMOTE.

First IEEE routine The first IEEE routine to execute after MINC is started invokes the IEEE_BUS_CLEAR routine automatically; this sets the REN line.

IEEE_BUS_CLEAR The IEEE_BUS_CLEAR routine clears and then sets the REN bus line.

LOCAL_INSTR The LOCAL_INSTR routine puts selected instruments in the local state. It does not affect the REN line.

LOCAL_LOCKOUT Pressing an instrument's return-to-local button puts the instrument in the local state. The LOCAL_LOCKOUT routine causes all instruments to ignore their return-to-local buttons. DISABLE_REMOTE cancels the effect of LOCAL_LOCKOUT, making the return-to-local buttons of all instruments operational.

SRQ_SUBROUTINE If DISABLE_REMOTE is used in a ser-

vice subroutine, your program should first check the state of the REN line with TEST_REMOTE. Before returning from the service subroutine, your program should return the REN line to the state it was in when the service subroutine was entered.

TEST_REMOTE The TEST_REMOTE routine reports whether the REN bus line is set or clear.

Restrictions

Certain MINC commands clear the REN line and then set it again if it was already set. This causes all instruments to enter the local state and makes all return-to-local buttons operative. (See "MINC Commands," page 29.)

Errors

?MINC-F-Invalid argument

?MINC-F-Conflict over control of the IEEE bus

Only MINC, the bus controller, can change the REN line.

Example

See the example for the LOCAL_INSTR routine.

ENABLE_PAR_POLL

Enable an Instrument's Parallel Poll Response

Operation

The ENABLE_PAR_POLL routine enables an instrument to respond to parallel polls, which are conducted by the PAR_POLL routine. (See "Parallel Polls," page 12.)

This routine tells the specified instruments to listen by sending the "unlisten" command and the appropriate "my listen address" and "my secondary address" commands. It then sends the "parallel poll configure" addressed command (which tells listeners to interpret commands of the form 011***** as parallel poll commands instead of as "my secondary address" commands), the "parallel poll enable" command for the condition and data line specified, and the "untalk" command (which tells instruments to stop interpreting commands of the form 011***** as parallel poll commands). (See "Commands: the ATN Line," page 20, and "Command Codes," page 22.)

Statement Form

ENABLE_PAR_POLL(condition,data-line,listeners)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
condition	numeric expression	0; 1	required argument
data-line	numeric expression	0 to 7	required argument
listeners	numeric expressions	0 to 30; 200 to 230	required argument

Example ENABLE_PAR_POLL(0,3,5)
Result From now on, when the routine PAR_POLL conducts a parallel poll, instrument 5 sets data line 3 if its status bit is clear (0).

Example ENABLE_PAR_POLL(1,3,5)
Result From now on, when the routine PAR_POLL conducts a parallel poll, instrument 5 sets data line 3 if its status bit is set (1).

Example ENABLE_PAR_POLL(1,7,5)
Result From now on, when the routine PAR_POLL conducts a parallel poll, instrument 5 sets data line 7 if its status bit is set.

Argument Descriptions

condition The instrument sets its data line during subsequent parallel polls if its status bit at the time of the poll is in the state specified by the condition argument.

Values	*Meaning*
0	The instrument sets its data line (specified in the data line argument) in response to subsequent parallel polls if its status bit is clear (0) at the time of the poll. If its status bit is set (1), the instrument does nothing.
1	The instrument sets its data line in response to subsequent parallel polls if its status bit is set (1) at the time of the poll. If its status bit is clear (0), the instrument does nothing.
Default:	required argument

The status bit is a bit in the instrument's interface which is clear or set to indicate some piece of information about the instrument. The meaning of the status bit is specific to that instrument and is explained in the user's guide for the instrument.

data line The data line of the IEEE bus on which the instrument responds to subsequent parallel polls.

Values:	0 to 7
Default:	required argument

The data-line argument specifies a data line of the IEEE bus. The instrument sets this line during subsequent parallel polls if its status bit at the time of the poll is in the state specified by the condition argument.

After your program has enabled an instrument's parallel poll response with ENABLE_PAR_POLL, it can change the data line and condition assigned to the instrument by using ENABLE_PAR_POLL again. Your program need not disable the parallel poll response first.

listeners The addresses of the instruments whose parallel poll responses are enabled.

Values:	0 to 30; 200 to 230
Default:	required argument

The listeners argument is a list of instrument addresses (0 to 30). Each address can be followed by a secondary address (200 to 230), though secondary addresses are not usually meaningful in this routine. Separate addresses with commas.

Related Routines

DISABLE_ALL_PAR_POLL, DISABLE_PAR_POLL The DISABLE_ALL_PAR_POLL and DISABLE_PAR_POLL routines disable the parallel poll response of instruments whose response has been enabled by ENABLE_PAR_POLL, causing these instruments not to respond to parallel polls.

PAR_POLL The PAR_POLL routine conducts a parallel poll. The ENABLE_PAR_POLL routine does not conduct a parallel poll; it only enables an instrument to respond when PAR_POLL conducts a poll.

Restrictions

Many instruments cannot respond to a parallel poll. Trying to enable the parallel poll response of an instrument that is incapable of responding to a parallel poll does not cause an error, but also does not enable the instrument to respond to parallel polls.

The parallel poll response of some instruments can be enabled and disabled only by local controls. Trying to enable the parallel poll response of such an instrument with ENABLE_PAR_POLL does not cause an error, but also does not enable its parallel poll response.

Each of the eight bits in the parallel poll response is clear unless one or more instruments sets it. By assigning only one instrument to each data line, your program knows whether each instrument did or did not set its line. If your program assigns more than one instrument to a single data line, then if that line is set in a parallel poll, your program does not know which instrument set it.

Errors

?MINC-F-Invalid argument

?MINC-F-Invalid instrument address

?MINC-F-Listener is not on the bus

The instrument you specified as a listener is either not on the bus, turned off, not designed to listen, set locally to "talk only," or not at the address specified.

Example

See also the example for the PAR_POLL routine.

Consider the following program fragment:

```
.
.
.
1100 DISABLE_ALL_PAR_POLL
```

```
1110 ENABLE_PAR_POLL(1,2,15)
1120 ENABLE_PAR_POLL(1,3,23)
1130 ENABLE_PAR_POLL(0,5,17)
1140 PAR_POLL(R1)
.
.
.
1530 DISABLE_PAR_POLL(23)
1540 PAR_POLL(R2)
.
.
.
```

Statement 1100 disables the parallel poll response of every instrument on the bus. Only the instruments whose parallel poll response is enabled in statements 1110, 1120, and 1130 respond to the parallel poll conducted in statement 1140. The bits of R1, the poll response, are shown in Figure 18.

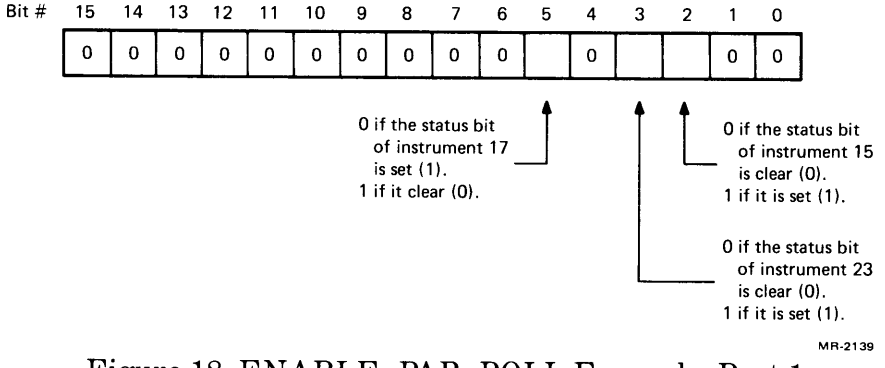

Figure 18. ENABLE_PAR_POLL Example, Part 1

Statement 1530 disables the parallel poll response of instrument 23. Assume that no parallel poll responses have been enabled or disabled since statement 1130. The bits of R2, the response to the poll conducted in statement 1540, are shown in Figure 19. Notice how the bits of the PAR_POLL argument shown in Figure 19 correspond to the data line values shown in Figure 9.

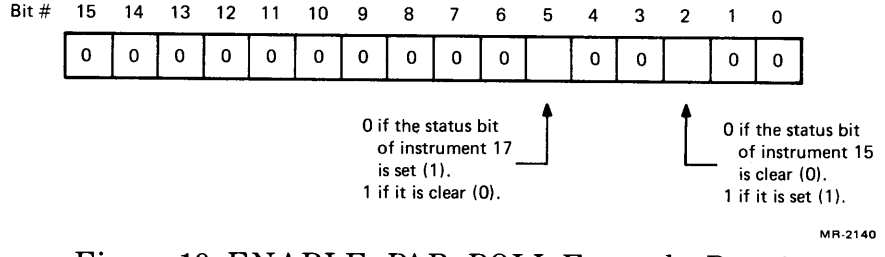

Figure 19. ENABLE_PAR_POLL Example, Part 2

ENABLE_REMOTE

Allow All Bus Instruments to Be in the Remote State

Operation

ENABLE_REMOTE sets the "remote enable" (REN) bus line. Whenever this line is clear, all instruments on the bus are in the local state, in which they use input information from their local controls. When the REN bus line is set, each instrument enters the remote state when your program specifies it as a listener in an IEEE routine. (See "Remote and Local," page 13.)

Statement Form

ENABLE_REMOTE

Argument	*Type of Argument*	*Valid Values*	*Default Value*
This routine has no arguments.			

Example ENABLE_REMOTE
Result Sets the REN bus line. Any instruments told to listen while this line is set enter the remote state.

Argument Descriptions

This routine has no arguments.

Related Routines

DISABLE_REMOTE The DISABLE_REMOTE routine clears the REN bus line. All instruments enter the local state.

First IEEE routine The first IEEE routine to execute after MINC is started invokes the IEEE_BUS_CLEAR routine automatically; this sets the REN line.

IEEE_BUS_CLEAR The IEEE_BUS_CLEAR routine clears and then sets the REN bus line.

SEND By specifying a string of length zero as the message to be sent in the SEND routine, you can tell instruments to listen without sending them a message. If the REN line is set, the instruments enter the remote state.

SRQ_SUBROUTINE If ENABLE_REMOTE is used in a service subroutine, your program should first check the state of the REN line with TEST_REMOTE. Before returning from the service subroutine, your program should return the REN line to the state it was in when the service subroutine was entered.

TEST_REMOTE The TEST_REMOTE routine reports whether the REN bus line is set or clear.

Restrictions

Certain MINC commands clear the REN line and then set it again if it was already set. This causes all instruments to enter the local state. (See "MINC Commands," page 29.)

Errors

?MINC-F-Invalid argument

?MINC-F-Conflict over control of the IEEE bus

Only MINC, the bus controller, can change the REN line.

Example

See the example for the LOCAL_INSTR routine.

IEEE_BUS_CLEAR

Clear the IEEE Bus

Operation

The IEEE_BUS_CLEAR routine clears the bus by returning the interface of every instrument on the bus to the clear state defined by the IEEE standard. (See "Clearing Interfaces," page 16.)

This routine sets the "interface clear" (IFC) bus line for approximately 125 microseconds. It clears and then sets the REN bus line.

Statement Form

IEEE_BUS_CLEAR

Argument	*Type of Argument*	*Valid Values*	*Default Value*
	This routine has no arguments.		

Example IEEE_BUS_CLEAR
Result Clear the instrument-independent part of all instruments on the bus.

Argument Descriptions

This routine has no arguments.

Related Routines

ALL_INSTR_CLEAR, INSTR_CLEAR IEEE_BUS_CLEAR does not affect the instrument-dependent parts of instruments. The ALL_INSTR_CLEAR and INSTR_CLEAR routines are complementary to IEEE_BUS_CLEAR and clear instrument-dependent parts of instruments but do not affect their interfaces.

DISABLE_REMOTE The DISABLE_REMOTE routine clears the REN bus line.

ENABLE_REMOTE The ENABLE_REMOTE routine sets the REN bus line.

First IEEE routine The first IEEE routine to execute after MINC is started invokes IEEE_BUS_CLEAR automatically.

LOCAL_LOOKOUT The IEEE_BUS_CLEAR routine cancels the effect of any previous LOCAL_LOCKOUT statements. After an IEEE_BUS_CLEAR statement, all return-to-local buttons of bus instruments are operative.

Restrictions

You can use IEEE_BUS_CLEAR at the beginning of your program to undo effects that the IEEE routines of previous programs have had on the bus.

Errors

?MINC-F-Invalid argument

?MINC-F-Conflict over control of the IEEE bus

Only MINC, the bus controller, can change the IFC line or the REN line.

Example

See the example for the SEND routine.

INSTR_CLEAR

Clear Selected Instruments

Operation

The INSTR_CLEAR routine clears the instrument-dependent part of specified instruments, returning each to a clear state defined by its manufacturer. (See "Clearing Instruments," page 16.)

This routine tells the specified instruments to listen by sending the "unlisten" command and the appropriate "my listen address" and "my secondary address" commands. It then sends the "selected device clear" addressed command. (See "Commands: the ATN Line," page 20, and "Command Codes," page 22.)

Statement Form

INSTR_CLEAR(listeners)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
listeners	numeric expressions	0 to 30; 200 to 230	required argument

Example INSTR_CLEAR(3)
Result Clear instrument 3.

Example INSTR_CLEAR(3,5,2)
Result Clear instruments 3, 5, and 2.

Argument Descriptions

Listeners The addresses of the instruments to be cleared.

Values: 0 to 30; 200 to 230

Default: required argument

The listeners argument is a list of instrument addresses (0 to 30). Each address can be followed by a secondary address (200 to 230), though secondary addresses are not usually meaningful in this routine. Separate addresses with commas.

Related Routines

ALL_INSTR_CLEAR The ALL_INSTR_CLEAR routine clears the instrument-dependent parts of instruments just as INSTR_CLEAR does, but ALL_INSTR_CLEAR affects all bus instruments, not just selected instruments.

IEEE_BUS_CLEAR INSTR_CLEAR does not affect the interfaces of instruments. IEEE_BUS_CLEAR is a complementary routine that clears the interfaces of instruments on the bus but does not affect their instrument-dependent parts.

Restrictions

Each instrument specified must be capable of being cleared by the bus controller. If an instrument's manufacturer has not implemented this part of the standard, then trying to clear the instrument with INSTR_CLEAR does not produce an error but also does not clear that instrument. Other instruments specified in the INSTR_CLEAR statement are still cleared.

Errors

?MINC-F-Invalid instrument address

?MINC-F-Listener is not on the bus

The instrument you specified as a listener is either not on the bus, turned off, not designed to listen, set locally to "talk only," or not at the address specified.

Example

See also the example for the LOCAL_INSTR routine.

Assume that your program has assigned the address of a voltmeter on the IEEE bus to the variable V%. The following program segment is a subroutine which takes a voltage reading on the 10 volt scale and stores the result in V$.

```
.
.
.
5200 INSTR_CLEAR(V%)
5210 SEND("10V",V%)
5220 RECEIVE(V$,,V%)
5230 RETURN
.
.
.
```

Because this subroutine might be called from several different places in the program, we don't know what state the voltmeter is in. Message routines might have changed the scale, polarity, leads used, or other parameters. Statement 5200 clears the voltmeter. Now that the voltmeter is in a known state, statement 5210 sets it to the 10 volt scale, and statement 5220 accepts the voltage reading.

INSTR_TIME_LIMIT

Set Time Allowed for Instrument Response

Operation

Use this routine to specify how long MINC will wait for an instrument to accept or send each character. (See "The Handshake Lines," page 26.)

Statement Form

INSTR_TIME_LIMIT(new-time-limit,old-time-limit)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
new-time-limit	numeric expression	0; 1 to 32,767	time limit not changed
old-time-limit	numeric variable name	0; 1 to 32,767	no value assigned

Example INSTR_TIME_LIMIT(300)
Result Set the new time limit to 300 clock ticks (five seconds if you have 60 Hz power, six seconds if you have 50 Hz power).

Example INSTR_TIME_LIMIT(,T)
Result Assign the number of clock ticks in the current time limit to T.

Example INSTR_TIME_LIMIT(,N)
INSTR_TIME_LIMIT(N+10)
Result First, N is set to the current time limit. Then, the time limit is reset to 10 clock ticks more than this.

Example INSTR_TIME_LIMIT(0)
Result There is no time limit. Wait indefinitely for any character to or from any instrument.

Argument Descriptions

new-time-limit The time limit for future characters.

Values	*Meaning*
0	Wait indefinitely for each character to be sent or received.
1 to 32,767	The maximum number of clock ticks that MINC waits for any one character to be sent or received. After waiting this long, MINC prints the "?MINC-F-Instrument time limit exceeded" error message and displays READY.

Default: Time limit remains unchanged.

The time limit is specified as a number of ticks of the systen clock. There are either 50 or 60 ticks per second, depending on whether your power is 50 or 60 Hz. The time limit is not instrument specific and applies to all command and message transmission on the bus.

When MINC is started, the time limit is 120 clock ticks.

old-time-limit The time limit before this INSTR_TIME_LIMIT statement. INSTR_TIME_LIMIT assigns a value to this argument.

Values	*Meaning*
0	There was no time limit. MINC was waiting indefinitely for each character to be sent or received.
1 to 32,767	The maximum number of clock ticks that MINC was waiting for any one character to be sent or received.
Default:	No value assigned. INSTR_TIME_LIMIT does not report the time limit.

Related Routines

RECEIVE, TRANSFER In the RECEIVE and TRANSFER routines, an instrument talks. If for any reason the instrument does not send a character within the time limit, MINC prints the "?MINC-F-Instrument time limit exceeded" error message and displays READY. Be sure that the instrument is on, addressed correctly, connected to the bus, and designed to talk.

SERIAL_POLL If an instrument does not respond to a serial poll within the time limit, MINC prints the "?MINC-F-Instrument time limit exceeded" error message and displays READY. Be sure that the instrument is on, addressed correctly, connected to the bus, and designed to respond to a serial poll.

Restrictions

You should rarely need this routine. If your program has an "?MINC-F-Instrument time limit exceeded" error, increasing the time limit might not help, because the time limit is 120 clock ticks when MINC is started and this is enough for most instruments. Before increasing the time limit, make sure that the instrument is connected to the bus, correctly addressed, and capable of the action you expect from it.

Use discretion in eliminating the time limit, since this causes the IEEE routines to wait indefinitely for each character. For ex-

ample, if you use RECEIVE to get data from instrument 4, and instrument 4 is not on the bus, then if there is no time limit, RECEIVE waits forever for the message, which will never be sent.

Errors

?MINC-F-Invalid argument

Example

A voltmeter on your IEEE bus automatically switches to the scale appropriate to the voltage being read. Because of this, the voltmeter can take up to 4 seconds to start sending a reading when you use the RECEIVE routine.

```
10 IEEE_BUS_CLEAR \ ALL_INSTR_CLEAR
20 V=13 \ F=60
30 INSTR_TIME_LIMIT(4*F)
40 RECEIVE(V$,,V) \ PRINT V$
50 STOP
```

Statement 10 clears the bus and the instruments on it. Statement 20 assigns 13, the address of the voltmeter, to the variable V and assigns 60, the frequency of your system clock, to the variable F. F should be set to 50 if you have a 50 Hz system clock. Statement 30 increases the time limit to four seconds before statement 40 accepts and prints a reading from the voltmeter.

LOCAL_INSTR

Put Selected Instruments in the Local State

Operation

LOCAL_INSTR causes specified instruments to enter the local state, in which the instruments use input information from their local controls instead of from the IEEE bus. (See "Remote and Local," page 13.)

This routine tells the specified instruments to listen by sending the "unlisten" command and the appropriate "my listen address" and "my secondary address" commands. It then sends the "go to local" addressed command. (See "Commands: the ATN Line," page 20, and "Command Codes," page 22.)

Statement Form

LOCAL_INSTR(listeners)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
listeners	numeric expressions	0 to 30; 200 to 230	required argument

Example LOCAL_INSTR(3,5)
Result Instruments 3 and 5 enter the local state.

Argument Descriptions

Listeners The addresses of the instruments to enter the local state.

Values: 0 to 30; 200 to 230

Default: required argument

The listeners argument is a list of instrument addresses (0 to 30). Each address can be followed by a secondary address (200 to 230), though secondary addresses are not usually meaningful in this routine. Separate addresses with commas.

Related Routines

DISABLE_REMOTE The DISABLE_REMOTE routine turns off the REN bus line, putting all bus instruments in the local state.

First IEEE routine The first IEEE routine to execute after MINC is started invokes IEEE_BUS_CLEAR automatically; this causes all instruments to enter the local state. Any listeners specified in this first routine, however, then enter the remote state.

IEEE_BUS_CLEAR The IEEE_BUS_CLEAR routine clears and then sets the REN bus line. This causes all instruments to enter the local state.

IEEE routines with listeners Any IEEE routine that requires a listener specification causes the instruments specified as listeners to enter the remote state if the REN bus line is set. This undoes the effect that any previous LOCAL_INSTR statement had on those instruments.

Restrictions

If a specified instrument does not have a local state, this routine has no effect on the instrument.

Errors

?MINC-F-Invalid instrument address

?MINC-F-Listener is not on the bus

The instrument you specified as a listener is either not on the bus, turned off, not designed to listen, set locally to "talk only," or not at the address specified.

Example

Instrument 5 on your IEEE bus is an ohmmeter, and instrument 27 is a signal generator. The following program measures the effects of voltage on a resistor. It measures the resistance, applies a voltage to the resistor for six hours, and then measures the resistance again. This program also allows you to use the ohmmeter through its front panel controls during the six hours.

```
10 IEEE_BUS_CLEAR \ ALL_INSTR_CLEAR
20 M=5 \ S=27
30 PRINT "Is a 5 ohm resistor in place";
40 INPUT A$ \ IF A$<>"YES" GO TO 30
50 LOCAL_LOCKOUT
60 SEND("10OHM",M)
70 RECEIVE(R1$,,M)
80 LOCAL_INSTR(M)
90 PRINT "Disconnect the ohmmeter leads."
100 PRINT "Are the leads disconnected";
110 INPUT A$ \ IF A$<>"YES" GO TO 100
120 PRINT "The ohmmeter can be used locally."
130 SEND("20V",S)
140 PAUSE("6::")
150 INSTR_CLEAR(S)
160 PRINT "Connect the ohmmeter leads."
170 PRINT "Are the leads connected";
180 INPUT A$ \ IF A$<>"YES" GO TO 170
190 INSTR_CLEAR(M)
200 SEND("10OHM",M)
```

```
210 RECEIVE(R2$,,M)
220 PRINT R1$; "ohms before"
230 PRINT R2$; "ohms after"
240 DISABLE_REMOTE \ ENABLE_REMOTE
250 STOP
```

Statement 10 clears the bus and the instruments on it. Statement 20 assigns 5, the address of the ohmmeter, to the variable M and 27, the address of the signal generator, to the variable S.

Assume that the signal generator has a return-to-local button. The LOCAL_LOCKOUT routine in statement 50 assures that no one will ruin the six-hour experiment by accidentally pressing this button.

The program now takes the first resistance reading. Statement 60 sets the ohmmeter to the 10 ohm scale, and statement 70 accepts the reading. The LOCAL_INSTR routine in statement 80 makes the ohmmeter available for local use. Without this statement, the ohmmeter could not be used locally. Even its return-to-local button would not enable its local controls because all such buttons were disabled in statement 50.

The signal generator starts outputting a 20 volt signal when statement 130 is executed. The PAUSE routine in statement 140 is a lab module routine and is described in Book 6. This statement causes MINC to wait six hours before continuing.

After six hours, clear the signal generator in statement 150. This stops its output. Then take the second resistance reading. Statement 190 clears the ohmmeter, canceling the effect of any local use. Since the INSTR_CLEAR routine specifies listeners, statement 190 also returns the ohmmeter to the remote state. Statement 200 sets the ohmmeter to the 10 ohm scale, and statement 210 accepts the reading.

Statement 240 turns the REN bus line off and then on again. This cancels the effect of the LOCAL_LOCKOUT routine so that the ohmmeter and signal generator can now be used locally. Because the REN line is on, they can also still be used remotely.

LOCAL_LOCKOUT

Disable Return-To-Local Buttons

Operation

An instrument on the IEEE bus can have a return-to-local button which causes it to enter the local state, in which the instrument uses input information from its local controls instead of from the IEEE bus. LOCAL_LOCKOUT causes the return-to-local buttons on all instruments on the bus to become inoperative. When pushed, they no longer activate the local controls of the instruments. (See "Remote and Local," page 13.)

This routine sends the "local lockout" universal command. (See "Commands: the ATN Line," page 20, and "Command Codes," page 22.)

Statement Form

LOCAL_LOCKOUT

Argument	*Type of Argument*	*Valid Values*	*Default Value*
	This routine has no arguments.		

Example LOCAL_LOCKOUT
Result Disable all return-to-local buttons of instruments on the bus.

Argument Descriptions

This routine has no arguments.

Related Routines

DISABLE_REMOTE The DISABLE_REMOTE routine clears the REN bus line. This undoes the effect of any previous LOCAL_LOCKOUT statement and puts all instruments on the bus in the local state. When your program calls ENABLE_REMOTE or IEEE_BUS_CLEAR to set the REN line again, the return-to-local buttons of all instruments on the bus are operational again.

IEEE_BUS_CLEAR The IEEE_BUS_CLEAR routine clears and then sets the REN bus line. This undoes the effect of any previous LOCAL_LOCKOUT, so that the return-to-local buttons of all instruments on the bus are again operational.

LOCAL_INSTR The LOCAL_INSTR routine puts selected instruments in the local state even if LOCAL_LOCKOUT has disabled their return-to-local buttons.

Restrictions

LOCAL_LOCKOUT has no effect on an instrument that does not have a return-to-local button. If an instrument has such a button, however, then according to the IEEE standard, that button must become inoperative when the LOCAL_LOCKOUT routine is executed.

Certain MINC commands clear the REN line and then set it again if it was already set. This causes all instruments to enter the local state and makes all return-to-local buttons operative. (See "MINC Commands," page 29.)

Errors

?MINC-F-Invalid argument

?MINC-F-Listener is not on the bus

No instrument is on the bus.

Example

See the example for the LOCAL_INSTR routine.

PAR_POLL

Conduct a Parallel Poll

Operation

The PAR_POLL routine conducts a parallel poll. Each instrument whose response has been previously enabled (either by local controls or by the ENABLE_PAR_POLL routine) reports the condition of its status bit on one of the bus's eight data lines. (See "Parallel Polls," page 12.)

The PAR_POLL routine sets both the "attention" (ATN) and the "identify" (IDY) bus lines and reads the poll response from the data lines. (See "Parallel Polls: IDY," page 21, and "Poll Results," page 25.)

Statement Form

PAR_POLL(poll-response)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
poll-response	numeric variable name	0 to 255	required argument

Example PAR_POLL(R)
Result Conduct a parallel poll. The response is assigned to the variable R.

Argument Descriptions

poll-response The response to the parallel poll. PAR_POLL assigns a value to the poll-response argument.

Values: 0 to 255

Default: required argument

Each responding instrument reports the state of its status bit, using the data line assigned to the instrument when its response was enabled. For each data line set in response to the poll, PAR_POLL sets the corresponding bit of the poll-response argument. (See "The Data Lines," page 22.)

Related Routines

DISABLE_ALL_PAR_POLL, DISABLE_PAR_POLL The DISABLE_ALL_PAR_POLL and DISABLE_PAR_POLL routines disable the parallel poll response of instruments whose response has been enabled by ENABLE_PAR_POLL, causing these instruments not to respond to parallel polls.

ENABLE_PAR_POLL The ENABLE_PAR_POLL routine enables selected instruments to respond to parallel polls. Only

instruments whose response has been previously enabled by ENABLE_PAR_POLL or by local controls respond to parallel polls.

SERIAL_POLL The SERIAL_POLL routine conducts a serial poll, whereas PAR_POLL conducts a parallel poll. Both types of polls ask instruments for information about their status, but there the similarity ends. An instrument reports eight bits of status information in a serial poll, and the bits of this *status byte* can be unrelated to the *status bit* reported in a parallel poll. SERIAL_POLL can poll only one instrument at a time, since an instrument must be told to talk in order to send its status byte. Unlike a parallel poll, a serial poll does not require any previous enabling of the instrument.

SRQ_SUBROUTINE, TEST_SRQ Some instruments use the status bit to indicate a request for service. This differs from requesting service by setting the SRQ bus line in the following ways:

1. An instrument that requests service by setting the SRQ line shares that line with all other bus instruments, so that MINC must conduct a serial poll to determine which instrument is requesting service. An instrument which indicates a request for service with its status bit, however, can be assigned its own dedicated data line on which to report its request for service during a parallel poll.

2. An instrument can set the SRQ bus line at any time and independently of MINC, and your program can designate a service subroutine to be entered automatically whenever an instrument requests service in this way. An instrument which uses its status bit to request service, however, can do nothing independently of MINC but must wait for your program to conduct a parallel poll before it can report its request for service.

TEST_BIT The TEST_BIT routine is described in Book 6 and tests whether a specified bit is clear or set. Use TEST_BIT after PAR_POLL to determine whether a specific bit of the parallel poll response is clear or set.

Restrictions

If no instrument has been assigned to a given data line, that line is clear when a parallel poll is conducted. If no instrument on the bus has been enabled to respond, every data line is clear when a parallel poll is conducted.

Each of the eight bits in the parallel poll response is clear unless one or more instruments sets it. By assigning only one instrument to each data line, your program knows whether each instrument did or did not set its line. If your program assigns more than one instrument to a single data line, then if that line is set in a parallel poll, your program does not know which instrument set it.

Errors

?MINC-F-Invalid argument

Example

See also the example for the ENABLE_PAR_POLL routine.

Assume that your IEEE bus has a signal generator whose address is 4 and a digital counter whose address is 7. The following program measures the number of counts detected by the counter during a three-second period in which a known voltage is being applied to a circuit. When the digital counter is triggered, it resets its count to zero and starts counting. It stops counting after the specified time and sets its status bit.

```
10 IEEE_BUS_CLEAR \ ALL_INSTR_CLEAR
20 C=7 \ G=4
30 SEND("10V",G)
40 SEND("3.0",C)
50 ENABLE_PAR_POLL(1,0,C)
60 TRIGGER_INSTR(C)
70 PAR_POLL(R)
80 TEST_BIT(0,S,R)
90 IF S=0 GO TO 70
100 RECEIVE(C$,,C)
110 PRINT C$; "counts detected."
120 STOP
```

Statement 10 clears the bus and the instruments on it. Statement 20 assigns 7, the address of the digital counter, to the variable C and assigns 4, the address of the signal generator, to the variable G. The program now tells the signal generator to output a 10 volt signal (statement 30) and tells the counter to count for three-second intervals (statement 40). Statement 50 tells the counter to set data line 0 in response to a parallel poll if its status bit is set.

The counter starts counting when it is triggered in statement 60. Statements 70, 80, and 90 form a loop that waits for the counter to report that its status bit is set. TEST_BIT is a lab module routine and is described in Book 6. After the counter sets its status bit, statements 100 and 110 accept and print the reading from the counter.

RECEIVE

Receive a Message from an Instrument

Operation

The RECEIVE routine receives and stores a message string from an instrument. (See "Receiving Messages," page 8.)

This routine tells the specified instrument to talk by sending the appropriate "my talk address" and "my secondary address" commands. MINC is a listener. If you specify listeners, this routine tells those instruments to listen also by sending the "unlisten" command and the appropriate "my listen address" and "my secondary address" commands. If you do not specify listeners, this routine sends the "unlisten" command so that MINC will be the only listener. MINC then clears the "attention" (ATN) bus line, allowing the talker to send message characters. (See "Commands: the ATN Line," page 20, "Command Codes," page 22, and "Messages Codes, page 22.)

Statement Form

RECEIVE(message,maximum-length,talker,listeners)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
message	string variable name	any string	required argument
maximum-length	numeric expression	0 to 255	255
talker	numeric expressions	0 to 30; 200 to 230	required argument
listeners	numeric expressions	0 to 30; 200 to 230	only MINC listens

Example RECEIVE(M$,,3)
Result Instrument 3 sends a message of up to 255 characters. MINC is the listener and stores the message string in M$.

Example RECEIVE(A1$,20,17,4)
Result Instrument 17 sends a message of up to 20 characters. MINC and instrument 4 are the listeners. MINC stores the message string in A1$.

Argument Descriptions

message The message string received from the instrument. RECEIVE assigns a value to the message argument.

Values: any string

Default: required argument

The meaning and format of the string is determined by the instrument's designer and is described in the user's guide for the

instrument. Numbers are usually sent as a string of characters and have to be converted to a numeric value by using the VAL function described in Book 3.

maximum-length The greatest number of characters MINC accepts from the talker.

Values: 0 to 255

Default: 255

MINC stops the talker from sending any more characters when the number of characters in this message reaches the limit specified in the maximum-length argument. MINC ends the message before this if the talker either sets the END bus line while sending a character or sends a terminator, a character MINC recognizes as an end of message indicator. Terminators are not stored as part of the message.

talker The address of the instrument that sends the message.

Values: 0 to 30; 200 to 230

Default: required argument

The talker argument is an instrument address (0 to 30), which can be followed by a secondary address (200 to 230). If a secondary address is specified, separate it from the instrument address with a comma.

listeners The addresses of the instruments that receive the message.

Values: 0 to 30; 200 to 230

Default: Only MINC listens.

The listeners argument is a list of instrument addresses (0 to 30). Each address can be followed by a secondary address (200 to 230). Separate addresses with commas.

MINC is a listener. Usually it is the only listener, but if you specify instruments in the listeners argument, those instruments also listen. Check the user's guide of any specified listener to be sure that the talker's message makes sense in the listener's vocabulary.

SEND, SEND_FRAGMENT The SEND and SEND_FRAGMENT routines send a string to an instrument.

MINC talks and the instrument listens.

SET_TERMINATORS After MINC is started, carriage return and line feed are terminators, characters MINC recognizes as end of message indicators. Some instruments, however, use other terminators. Use the SET_TERMINATORS routine to specify terminators.

TRANSFER The TRANSFER routine supervises a message transfer from one instrument to another. It is similar to RECEIVE except that MINC does not store the message.

TRIGGER_INSTR The TRIGGER_INSTR routine is sometimes used in conjunction with the RECEIVE routine. TRIGGER_INSTR might tell an instrument to take a reading; the instruments report the results of that reading in the next RECEIVE routine.

Restrictions

Some instruments send a carriage return and a line feed at the end of each message. Normally MINC recognizes each of these characters as a terminator. When a RECEIVE statement accepts a message from such an instrument, MINC ends the message when the carriage return is sent. The next RECEIVE statement from that instrument accepts the line feed, which immediately ends that message. Therefore each full message from the instrument requires two RECEIVE statements if the message terminators have not been changed with the SET_TERMINATORS routine.

MINC can receive messages in fragments, using more than one RECEIVE command, by specifying the length of each fragment in the message-length argument. MINC stops the instrument in the middle of its message if the message length specified is shorter than the length of the instrument's actual message. The instrument continues the message from that point if it is told to talk again.

Errors

?MINC-F-Invalid argument

?MINC-F-Invalid instrument address

?MINC-F-Not enough space for the string

There is not enough workspace to store the message string from the talker.

?MINC-W-Same instrument specified as talker and listener

MINC prints this warning but allows the message to be transmitted.

?MINC-F-Instrument time limit exceeded

The talker specified is either not on the bus, turned off, not designed to talk, set locally to "listen only," or not at the address specified. Some instruments need a longer time limit, which you can specify with the INSTR_TIME_LIMIT routine.

?MINC-F-Listener is not on the bus

No instrument is on the bus.

See also the example for the TRIGGER_INSTR routine.

Example

Instrument 4 on your IEEE bus reports readings in the following format:

v±d.dddE±dd←↓

where

- v is the letter "V" if the reading is valid and the letter "O" if the reading is off scale
- ± is either a "+" or a "-" character
- d is a digit, a character between "0" and "9"
- E is the letter "E"
- ← is the "return" character
- ↓ is the "line feed" character

The following program takes ten readings from the instrument and prints the average.

```
10 IEEE_BUS_CLEAR \ ALL_INSTR_CLEAR
20 I%=4 \ R=0
30 SET_TERMINATORS(10)
40 FOR N=0 TO 9
50 RECEIVE(V$,1,I%)
60 RECEIVE(R$,,I%)
70 IF V$="O" THEN PRINT "Data overflow" \ STOP
80 R$=SEG$(R$,1,LEN(R$)-1)
90 R=R+VAL(R$)
100 NEXT N
110 R=R/10 \ PRINT "The average is"; R
120 STOP
```

Statement 10 clears the IEEE bus and all instruments on it. Statement 20 assigns the instrument address to I% and zeroes the sum of the readings.

This instrument sends both a carriage return and a line feed at the end of each message. The SET_TERMINATORS routine in statement 30 makes only line feed a terminator so that the messages will not end prematurely when the carriage return character is sent.

The string V$ in statement 50 is the first character of the message. The string R$ in statement 60 is the remainder of the message. If V$ is an "O" character, the reading is invalid so we stop the program.

Statement 80 deletes the last character (the carriage return) from R$. The VAL function in statement 90 then converts the remaining string into a number which is added to the total, R (see Book 3). After ten readings, statement 110 calculates and prints the average.

SEND

Send a Message to an Instrument

Operation

The SEND routine sends a message string to one or more instruments on the IEEE bus. (See "Sending Messages," page 7.)

MINC itself is the talker, so this routine first sends an UNTALK command, which tells the previous talker not to talk anymore. Then it tells the specified instruments to listen by sending the "unlisten" command and the appropriate "my listen address" and "my secondary address" commands. MINC then clears the "attention" (ATN) bus line and sends the string one character at a time, setting the END bus line while it sends the last character of the string. (See "Commands: the ATN Line," page 20, "Command Codes," page 22, and "Message Codes," page 22.)

Statement Form

SEND(message,listeners)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
message	string expression	any string	required argument
listeners	numeric expressions	0 to 30; 200 to 230	required argument

Example SEND("R1F3", 17).
Result MINC (the talker) sends the string "R1F3" to instrument 17 (the listener). It sets the END bus line while sending the "3."

Example SEND(M$,5,214)
Result MINC sends the string M$ to secondary address 14 of instrument 5. It sets the END bus line while sending the last character of the string.

Argument Descriptions

message The string that MINC sends.

Values: any string

Default: required argument

Each instrument that can listen has a vocabulary of characters that are meaningful to it. This vocabulary is instrument specific and not defined by the IEEE standard. The message you send to an instrument must make sense in terms of the instrument's vocabulary. If it does not, the instrument might ignore the message, or it might do something other than what you intended.

listeners The addresses of the instruments that are to receive the message.

Values: 0 to 30; 200 to 230

Default: required argument

The listeners argument is a list of instrument addresses (0 to 30). Each address may be followed by a secondary address (200 to 230). Separate addresses with commas.

Related Routines

ENABLE_REMOTE By specifying a string of length zero in the message argument of the SEND routine, you can tell instruments to listen without sending them a message. If the REN line is set, the instruments enter the remote state.

RECEIVE The RECEIVE routine receives a string from an instrument. MINC listens and the instrument talks.

SEND_FRAGMENT The SEND_FRAGMENT routine also sends a string to an instrument. It is identical to the SEND routine except that it does not set the END bus line while sending the last character of the string. This allows MINC to send messages longer than 255 characters, which is the maximum length of a MINC string.

TEST_LISTENERS If a SEND statement specifies a listener address which does not correspond to any instrument on the bus, MINC prints the "?MINC-F-Listener is not on the bus" error message and displays READY. The TEST_LISTENERS routine checks for this error condition without stopping your program.

Restrictions

Some instruments expect each message to end with a special terminating character, such as a carriage return, and do not act on a message until the terminating character is sent. The user's guide for a particular instrument tells you whether or not that instrument requires a terminator. If the instrument does require a terminator, you must explicitly specify the terminator as part of the message string. Use the CHR$ function described in Book 3 when specifying a non-printing character.

The SEND routine is not discriminating: it can send any string to any listener. In order to use this routine effectively, you must know the vocabulary of every instrument you send a message to; this information is in the user's guide for each instrument.

Errors

?MINC-F-Invalid argument

?MINC-F-Invalid instrument address

?MINC-F-Listener is not on the bus

The instrument you specified as a listener is either not on the bus, turned off, not designed to listen, set locally to "talk only," or not at the address specified.

Example

Instrument 7 on your IEEE bus is a signal generator. Its frequency is programmed by sending it the character "F" followed by a number, and it requires that messages to it end with the "return" character (ASCII value 13). For example, the string "F7500" plus a "return" character causes it to generate a signal at 7500 cycles per second. The following program asks you for a frequency and then generates that signal.

```
10 IEEE_BUS_CLEAR \ ALL_INSTR_CLEAR
20 G=7
30 PRINT "What frequency";
40 INPUT F \ IF F <0 GO TO 30
50 SEND("F"+STR$(F)+CHR$(13),G)
60 STOP
```

Statement 10 clears the bus and all the instruments on it. Statements 30 and 40 ask you for the frequency and test that the value is valid.

Statement 50 sends the specified concatenated string to instrument 7. The STR$ function converts a number into its string representation, and the CHR$ function converts an ASCII value into a one-character string (see Book 3).

SEND_FRAGMENT

Send a Message Fragment to an Instrument

Operation

The SEND_FRAGMENT routine sends a message fragment to one or more instruments on the IEEE bus. It should be used to send part of a message if the entire message is longer than 255 characters, the maximum length of a MINC string. (See "Sending Messages," page 7, and "Fragmented Messages," page 21.)

MINC itself is the talker, so it first sends an "untalk" command, which tells the previous talker not to talk any more. Then it tells the specified instruments to listen by sending the "unlisten" command and the appropriate "my listen address" and "my secondary address" commands. MINC then clears the "attention" (ATN) bus line and sends the string one character at a time. Unlike the SEND routine, SEND_FRAGMENT does not set the END bus line while it sends the last character of the string. (See "Commands: the ATN Line," page 20, "Command Codes," page 22, and "Message Codes," page 22.

Statement Form

SEND_FRAGMENT(message-fragment,listeners)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
message-fragment	string expression	any string	required argument
listeners	numeric expressions	0 to 30; 200 to 230	required argument

Example SEND_FRAGMENT(F$,3)
Result Send the string F$ to instrument 3. This string is only part of the message to instrument 3. The last part will be sent with the SEND routine.

Argument Descriptions

message-fragment The string that MINC sends.

Values: any string

Default: required argument

Each instrument that can listen has a vocabulary of characters that are meaningful to it. This vocabulary is instrument specific and not defined by the IEEE standard. The message you send to an instrument must make sense in terms of the instrument's vocabulary. If it does not, the instrument might ignore the message, or it might do something other than what you intended.

This routine does not set the END bus line when it sends the last

character of the string. Use SEND_FRAGMENT to send parts of messages that are longer than 255 characters, the maximum MINC string length. Use SEND_FRAGMENT to send all parts of such a message except the last, which should be sent with SEND.

listeners The addresses of the instruments that are to receive the string.

Values: 0 to 30; 200 to 230

Default: required argument

The listeners argument is a list of instrument addresses (0 to 30). Each address may be followed by a secondary address (200 to 230). Separate addresses with commas.

Related Routines

RECEIVE The RECEIVE routine receives a string from an instrument. MINC listens and the instrument talks.

SEND The SEND routine sends a string to an instrument. It is identical to the SEND_FRAGMENT routine except that it sets the END bus line while sending the last character of the string.

TEST_LISTENERS If a SEND_FRAGMENT statement specifies a listener address which does not correspond to any instrument on the bus, MINC prints the "?MINC-F-Listener is not on the bus" error message and displays READY. The TEST_LISTENERS routine checks for this error condition without stopping your program.

Restrictions

The END line indicates to many instruments that the character being sent is the last character in the message, and some instruments require that this line be set before they act on a message sent to them.

The SEND_FRAGMENT routine is not discriminating: it can send any string to any listener. In order to use this routine effectively, you must know the vocabulary of the instruments you are sending messages to.

Errors

?MINC-F-Invalid argument

?MINC-F-Invalid instrument address

?MINC-F-Listener is not on the bus

The instrument you specified as a listener is either not on the

bus, turned off, not designed to listen, set locally to "talk only," or not at the address specified.

Example

None

SERIAL_POLL
Conduct a Serial Poll

Operation

The SERIAL_POLL routine conducts a serial poll. It tells your program which instrument is requesting service and the value of its status byte. (See "Status," page 9.)

This routine sends the "serial poll enable" universal command, which puts bus instruments into serial poll mode. In this mode, any instrument that is told to talk sends a one-character message, its status byte. To poll each instrument, MINC tells the instrument to talk by sending the appropriate "my talk address" and "my secondary address" commands. MINC then clears the "attention" (ATN) bus line, and the instrument sends its status byte. When an instrument's status byte indicates that it is requesting service, no further instruments are polled and this routine sends the "serial poll disable" universal command to take bus instruments out of serial poll mode. (See "Commands: the ATN Line," page 20, and "The Data Lines," page 22.)

Statement Form

SERIAL_POLL(status,index,talkers)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
status	numeric variable name	0 to 255	no value assigned
index	numeric variable name	0; ⩾1	no value assigned
talkers	numeric expressions	0 to 30; 200 to 230	required argument

Example
SERIAL_POLL(D%,,4)
Result
Conduct a serial poll of instrument 4 and put its status byte in the variable D%.

Example
SERIAL_POLL(,I,4)
Result
Conduct a serial poll of instrument 4 to determine whether or not it is requesting service. I is assigned the value 1 if it is requesting service, 0 if it is not.

Example
SERIAL_POLL(S,I,5,7,3)
Result
Conduct a serial poll of instruments 5, 7, and 3. If instrument 5 is requesting service, I is 1 and S is the status byte of instrument 5. If instrument 5 is not requesting service but instrument 7 is, I is 2 and S is the status byte of instrument 7. If neither instrument 5 nor 7 is requesting service but instrument 3 is, I is 3 and S is the

status byte of instrument 3. If none of the instruments 5, 7, and 3 is requesting service, I is 0 and S is the status byte of the last instrument, instrument 3.

Example SERIAL_POLL(,I%,4,7,2,19,3)

Result Conduct a serial poll of instruments 4, 7, 2, 19, and 3. I% is assigned the value 0 if none of these instruments is requesting service; 1 if instrument 4 is requesting service; 2 if 4 is not requesting service but 7 is; and so forth.

Argument Descriptions

status The status byte of the last instrument polled, as indicated by the index argument. SERIAL_POLL assigns a value to the status argument.

Values: 0 to 255

Default: No value assigned. SERIAL_POLL does not report the status byte.

The status argument is the status byte of the instrument indicated by the index argument. If the index argument is 0, the status argument is the status byte of the last instrument specified in the talkers argument. As shown in Figure 20, bit 6 of the status argument always indicates whether or not the instrument was requesting service. Bits 0 through 5 and bit 7 indicate other aspects of the instrument's status. Notice how the bits of the status argument shown in Figure 20 correspond to the data lines of the serial poll response shown in Figure 8, page 11.

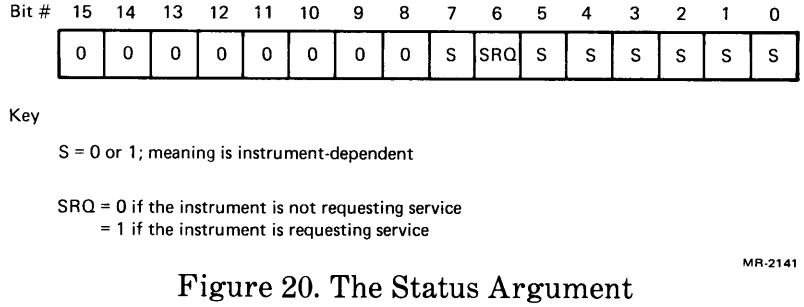

Figure 20. The Status Argument

index The index argument indicates which instrument was requesting service. SERIAL_POLL assigns a value to this argument.

Values	*Meaning*
0	No instrument was requesting service.

⩾1	An index to the instrument requesting service.
Default:	No value assigned. SERIAL_POLL does not report which instrument was requesting service.

As it polls each instrument, MINC checks data line 6 of the instrument's response. If line 6 is clear, the instrument was not requesting service, so MINC polls the next instrument in the list. If line 6 is set, however, the instrument was requesting service, so MINC polls no other instruments. The index argument is 1 if the first instrument listed was requesting service; 2 if the first instrument was not requesting service but the second one was; 5 if the fifth instrument was requesting service but none of the first 4 were; and so on. If none of the instruments in the list of talkers was requesting service, the index argument is 0.

More than one instrument can request service at the same time. When this happens, the order of the instruments in the talkers argument represents the priority of the service requests. An instrument whose address is earlier in the list has its service request recognized before an instrument whose address is later in the list. For example, suppose instruments 5 and 7 are requesting service. Your program uses SERIAL_POLL to determine the source of the service request. If instrument 7 is listed first in the SERIAL_POLL statement, its service request is recognized. Instrument 5 is not polled and so it continues to request service. When the SERIAL_POLL statement is executed again, instrument 7 reports that it is no longer requesting service, so instrument 5 is polled. Its service request is now recognized.

Some instruments do not clear bit 6 of their status byte when they are serially polled, even though they are no longer requesting service. These instruments do not clear bit 6 until MINC actually services them. For example, a voltmeter might request service when it has a data reading to report. Though it would stop requesting service when serially polled, the voltmeter might not clear bit 6 of its status byte until it reports the reading in a RECEIVE statement.

talkers The addresses of the instruments to be polled.

Values:	0 to 30; 200 to 230
Default:	required argument

The talkers argument is a list of instrument addresses (0 to 30). Each address can be followed by a secondary address (200 to

230), though secondary addresses are not usually meaningful in this routine. Separate addresses with commas.

At least one talker must be specified. If your program is using this routine to determine the source of a service request, be sure to include in the list of talkers all instruments on the bus that can request service. This assures that SERIAL_POLL will find the source of the service request.

Related Routines

PAR_POLL The PAR_POLL routine conducts a parallel poll, whereas SERIAL_POLL conducts a serial poll. Both types of polls ask instruments for information about their status, but there the similarity ends. An instrument reports only one bit of status information in a parallel poll, and this *status bit* can be unrelated to any of the bits in the *status byte* reported in a serial poll. Instruments must be enabled in advance to respond to parallel polls. When PAR_POLL conducts a parallel poll, all enabled instruments respond at the same time, and none of these instruments has to be told to talk.

SRQ_SUBROUTINE, TEST_SRQ The SRQ_SUBROUTINE and TEST_SRQ routines detect service requests. After a service request is detected, use SERIAL_POLL to identify the source of the request. Conducting a serial poll of the instrument requesting service frees the service request (SRQ) bus line for future use.

TEST_BIT The TEST_BIT routine is described in Book 6 and tests whether a specified bit is clear or set. A status byte has eight bits, each of which may indicate specific information about the instrument's status. Use TEST_BIT after SERIAL_POLL to determine whether a specific bit of the status byte is clear or set.

Restrictions

The instrument being polled must be capable of responding to a serial poll. Any instrument that can request service can respond to a serial poll.

Errors

?MINC-F-Invalid argument

?MINC-F-Invalid instrument address

?MINC-F-Instrument time limit exceeded

The instrument specified is either not on the bus, not turned on, or unable to respond to a serial poll.

?MINC-F-Listener is not on the bus

No instrument is on the bus.

Examples

See also the example for the SRQ_SUBROUTINE routine.

Suppose that a plotter on your IEEE bus can respond to serial polls and reports the information shown in Figure 21 in its status byte. Bit 2 is set if the plotter is out of ink, bit 5 is set if it is out of paper, and bit 6 is set if it is requesting service. The other bits are always clear.

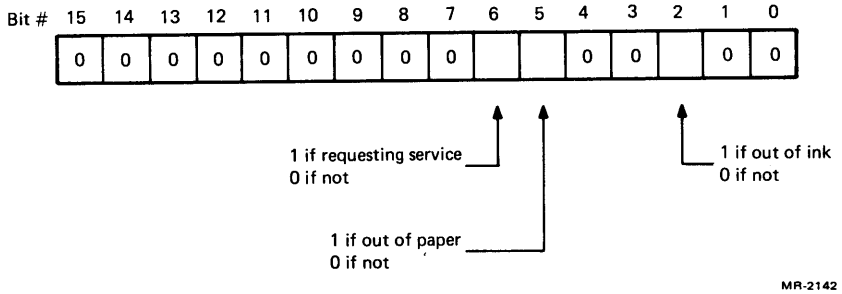

Figure 21. SERIAL_POLL Example

The following program checks whether or not the plotter is out of paper.

```
10 IEEE_BUS_CLEAR \ ALL_INSTR_CLEAR
20 P%=6
30 SERIAL_POLL (S,,P%)
40 TEST_BIT(5,P,S)
50 IF P=0 GO TO 100
60 PRINT "Please put paper in the plotter."
70 PRINT "Have you put in the paper";
80 INPUT A$ \ IF A$<>"YES" GO TO 70
90 GO TO 30
100 PRINT "The plotter has paper."
110 STOP
```

Statement 10 clears the IEEE bus and the instruments on it. Statement 20 assigns 6, the address of the plotter, to the variable P%. After statement 30 gets the status byte of the plotter, statement 40 checks bit 5 with TEST_BIT, a lab module routine described in Book 6. If the plotter has enough paper, bit 5 is clear so P is 0. If bit 5 is set, P is not zero so statements 60 through 90 are executed.

SET_TERMINATORS
Specify Terminating Characters

Operation

This routine specifies the characters that the routines RECEIVE and TRANSFER recognize as message terminators. (See "Receiving Messages," page 8.)

Statement Form

SET_TERMINATORS (terminators)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
terminators	numeric or string expression	0 to 255; any one-character string	no characters are terminators

Example SET_TERMINATORS(10)
Result Only the line feed character (ASCII value 10) is recognized by RECEIVE and TRANSFER as terminators.

Example SET_TERMINATORS('A','B',10,13)
Result Characters 'A', 'B', line feed (ASCII value 10), and carriage return (ASCII value 13) are recognized by RECEIVE and TRANSFER as terminators.

Example SET_TERMINATORS()
Result No characters are recognized by RECEIVE and TRANSFER as terminators.

Argument Descriptions

terminators A list of up to 4 message terminating characters.

Values	*Meaning*
0 to 255	The ASCII value of a character that is to be a terminator.
any one-character string expression	A character that is to be a terminator.
Default:	RECEIVE and TRANSFER do not recognize any character as a terminator.

Up to four terminators can be specified. These terminators override any previous terminators. After MINC is started, carriage return and line feed are the terminators.

Related Routines

RECEIVE, TRANSFER RECEIVE and TRANSFER are the only routines that use the list of terminators specified in SET_TERMINATORS. Each time the talker sends a character, these routines check that character against the list of terminators. If the character is a terminator, they end the message. RECEIVE does not store the terminator as part of the message.

SEND, SEND_FRAGMENT Some instruments require that a special terminating character be sent *to* them to end a message. This terminating character, however, must be explicitly specified in SEND. The terminators specified in SET_TERMINATORS apply only to characters that instruments themselves send.

SRQ_SUBROUTINE, CONTINUE, SCHEDULE, and SCHMITT SRQ_SUBROUTINE, CONTINUE, SCHEDULE, and SCHMITT are the MINC routines that can schedule service subroutines. When a service subroutine executes a RETURN statement, MINC makes the list of message terminators the same as it was when the subroutine was invoked. Thus, if SET_TERMINATORS is used in a service subroutine, the new terminators are used only for the duration of the subroutine. SET_TERMINATORS is the only IEEE bus routine whose effect is automatically reset at the end of a service subroutine. CONTINUE, SCHEDULE, and SCHMITT are lab module routines and are discussed in Book 6.

Restrictions

If a character is a terminator, it is recognized as such regardless of which instrument sends it. If one instrument uses a character as a terminator, and another uses it as part of a message, then the latter's message could be prematurely ended if it sends the character recognized by MINC as a terminator. Using SET_TERMINATORS to eliminate this character as a terminator would allow the message to conclude properly.

Some instruments send a carriage return and a line feed at the end of each message. Normally MINC recognizes each of these characters as a terminator. When a RECEIVE statement accepts a message from such an instrument, MINC ends the message when the carriage return is sent. The next RECEIVE statement from that instrument accepts the line feed, which immediately ends that message. Therefore each full message from the instrument requires two RECEIVE statements if the message terminators have not been changed with the SET_TERMINATORS routine.

MINC recognizes no more than four terminators at any time.

Errors

?MINC-F-Invalid argument

Example

See the example for the RECEIVE routine.

SRQ_SUBROUTINE

Designate an SRQ Service Subroutine

Operation

With this routine, your program can designate a service subroutine that will be entered automatically whenever any instrument requests service by setting the SRQ bus line. (See "Status," page 9.)

Statement Form

SRQ_SUBROUTINE(subroutine)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
subroutine	numeric expression	0; 1 to 32,767	0

Example SRQ_SUBROUTINE(2300)
Result When a service request occurs, start executing the service subroutine that begins at statement number 2300.

Example SRQ_SUBROUTINE(S%)
Result Designate an SRQ subroutine whose beginning statement number is the value of the variable S%. When a service request occurs, this service subroutine is invoked.

Example SRQ_SUBROUTINE()
Result Cancel the current SRQ service subroutine.

Argument Descriptions

subroutine The number of the first statement in the SRQ service subroutine.

Values	*Meaning*
1 to 32,767	Statement number of a service subroutine.
0	Cancel the current SRQ service subroutine. When a service request occurs, MINC does not automatically invoke a service subroutine.

Default: 0

SRQ_SUBROUTINE designates a service subroutine by the statement number of the first statement in the subroutine. When an instrument sets the SRQ bus line, MINC transfers program control to the service subroutine as soon as the current statement finishes executing. After the service subroutine, the

program returns to the statement following the one it was executing when the service request occurred.

Related Routines

DISABLE_REMOTE, ENABLE_REMOTE, and TEST_REMOTE If your program uses the DISABLE_REMOTE routine or the ENABLE_REMOTE routine in a service subroutine, then it should first check the state of the REN line with TEST_REMOTE. Before returning from the service subroutine, your program should return to REN line to the state it was in when the subroutine was entered.

SERIAL_POLL The service subroutine designated by SRQ_SUBROUTINE should include a SERIAL_POLL statement which polls all instruments that can request service. This tells your program which instrument is requesting service and causes that instrument to stop requesting service. The service subroutine can then execute different statements depending on which instrument is requesting service.

SET_TERMINATORS When a service subroutine executes a RETURN statement, MINC makes the list of message terminators the same as it was when the subroutine was invoked. Thus, if SET_TERMINATORS is used in a service subroutine, the new terminators are used only for the duration of the subroutine. SET_TERMINATORS is the only IEEE bus routine whose effect is automatically reset at the end of a service subroutine.

TEST_SRQ Instead of using SRQ_SUBROUTINE to designate a service subroutine to handle service requests, your program can use TEST_SRQ periodically to test the SRQ bus line. This method of detecting service requests should only be used when fast response is not necessary.

CONTINUE, SCHEDULE, and SCHMITT CONTINUE, SCHEDULE, and SCHMITT are lab module routines that designate service subroutines, which are just like the SRQ service subroutine except that different conditions cause them to be executed. They are described in Book 6.

Restrictions

All service subroutines are canceled when MINC displays READY. This makes them useless in immediate mode.

More than one instrument can request service at the same time. While the SRQ subroutine is executing, new SRQs are ignored. When the RETURN statement is executed, MINC checks the SRQ line. If the line is clear, then the main program resumes. If

the line is set, however, then MINC checks whether or not the SRQ subroutine has serially polled any instrument requesting service. If it has, then the SRQ line is set because one or more other instruments are also requesting service, so MINC does not return to the main program but instead executes the SRQ subroutine again. If it has not, then the SRQ line is set because the instrument setting the line has not stopped requesting service. This situation could cause the SRQ subroutine to be executed over and over again for the same service request. For this reason, MINC automatically cancels the SRQ subroutine and prints the "?MINC-W-SRQ subroutine cancelled because no serial poll done" error message.

The resequencing command, RESEQ, resequences normal program statement numbers. It does not change subroutine statement numbers specified in SRQ_SUBROUTINE statements. When you resequence a program, you have to determine the new statement number of the service subroutine and change the subroutine argument to that number in the SRQ_SUBROUTINE statement. For this reason, it is more convenient to use a variable name for the subroutine argument. Put the variable assignment statement and an explanatory remark at the beginning of the program where you can locate it quickly. Then, each time you resequence the program, assign new values to the statement number variables and save searching for all occurrences of SRQ_SUBROUTINE statements.

Errors

?MINC-F-Invalid argument

?MINC-W-SRQ subroutine canceled because no serial poll done

?MINC-W-SRQ service subroutine has been canceled

?MINC-F-Could not find service subroutine #### requested

Example

The following program uses lab module routines (described in Book 6) to control data acquisition for an experiment involving radioisotopes. Your IEEE bus has only four instruments: two identical radiation detectors, each of which issues a service request if the radiation level of the room it is in rises above a certain level, and two identical alarms, each of which sounds when triggered by MINC. The radioisotope is used at the start of the experiment but is then transferred to another room for storage. One detector and one alarm are in each room. The program sounds a room's alarm immediately if the radiation in that room reaches a hazardous level. If the radiation leak is in the experiment room, the program stops the experiment.

```
10 IEEE_BUS_CLEAR \ ALL_INSTR_CLEAR
20 D1=11 \ A1=12 \ D2=21 \ A2=22 \ S=5000
30 SRQ_SUBROUTINE(S)
.
.
.
rest of program acquires experiment data
.
.
.
5000 SERIAL_POLL(,I%,D1,D2)
5010 IF I%=0 THEN PRINT "SRQ error" \ RETURN
5020 ON I% GO TO 5100,5200
.
.
.
5100 TRIGGER_INSTR(A1)
5110 STOP
5200 TRIGGER_INSTR(A2)
5210 RETURN
```

Statement 10 clears the IEEE bus and the instruments on it. Statement 20 contains values for the addresses of the detectors in the experiment room (D1) and the storage room (D2) and the alarms in the experiment room (A1) and the storage room (A2). Statement 30 designates 5000, the value of S, as the beginning of the SRQ service subroutine.

If a detector requests service, the service subroutine is automatically executed. Statement 5000 conducts a serial poll to determine which detector is requesting service. In statement 5010, I% should never be zero since the two detectors are the only instruments on the bus that can request service. Include this statement, however, in case some other instrument is added to the bus later. If I% were 0, statement 5020 would cause a fatal error.

Statement 5020 branches to different points depending on the value of I%. If the detector in the experiment room was requesting service, I% is 1 and statements 5100 and 5110 are executed. These statements sound the alarm in the experiment room and stop the experiment. If the detector in the storage room was requesting service, I% is 2 and statements 5200 and 5210 are executed. These statements sound the alarm in the storage room, but allow the experiment to continue.

Notice in statement 5000 that the detector in the experiment room has the higher priority. If both detectors request service simultaneously, I% is 1.

TEST_LISTENERS

Test for the Presence of Listeners

Operation

This routine tests whether any of the listeners specified are on the bus.

TEST_LISTENERS tries to send a line feed character with the END line set. If the acceptor handshake is not detected, no instrument is listening. (See "The Handshake Lines," page 26.)

Statement Form

TEST_LISTENERS(condition, listeners)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
condition	numeric variable name	0; -1	required argument
listeners	numeric expressions	0 to 30; 200 to 230	required argument

Example TEST_LISTENERS(C,3)
Result C is -1 if instrument 3 is listening, 0 if it is not.

Example TEST_LISTENERS(A%,3,6,7)
Result A% is -1 if any of the instruments 3, 6, and 7 are listening, 0 if none of them are.

Argument Descriptions

condition Reports whether or not any of the instruments specified are listening. TEST_LISTENERS assigns a value to the condition argument.

Values	*Meaning*
0	None of the instruments are listening.
-1	At least one of the instruments is listening.
Default:	required argument

An instrument may not be listening for any of the following reasons:

1. The instrument is not connected to the bus.
2. The instrument's power is not turned on. (Never turn an instrument's power on while MINC is running!)
3. The instrument has an address different from the one you specified.

4. The instrument is not designed to listen.

5. Some instruments have a switch which can be set to "talk only." This switch can prevent the instrument from listening.

listeners The addresses to be tested.

Values: 0 to 30; 200 to 230

Default: required argument

The listeners argument is a list of instrument addresses (0 to 30). Each address may be followed by a secondary address (200 to 230). Separate addresses with commas.

Related Routines

SEND, SEND_FRAGMENT TEST_LISTENERS should be used before SEND or SEND_FRAGMENT if the listener might not be on the bus. SEND and SEND_FRAGMENT produce a fatal error if the listener is not on the bus.

Restrictions

TEST_LISTENERS sends a one character message, a line feed. Check the user's guide for each instrument to be sure this has no undesired effect.

Errors

?MINC-F-Invalid argument

?MINC-F-Invalid instrument address

?MINC-F-Listener is not on the bus

No instrument is on the bus.

Example

None

TEST_REMOTE

Test the Remote Enable (REN) Bus Line

Operation

TEST_REMOTE tells your program whether or not the remote-enable (REN) line of the IEEE bus is set. (See "Remote and Local," page 13.)

Statement Form

TEST_REMOTE(condition)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
condition	numeric variable name	0; -1	required argument

Example TEST_REMOTE(C)
Result Tests the REN bus line. C is -1 if the line is set, 0 if it is clear.

Argument Descriptions

condition The condition of the REN bus line. TEST_REMOTE assigns a value to the condition argument.

Values	*Meaning*
0	The REN line is clear.
-1	The REN line is set.
Default:	required argument

This routine does not change the state of the REN line.

Related Routines

DISABLE_REMOTE The DISABLE_REMOTE routine clears the REN bus line.

ENABLE_REMOTE The ENABLE_REMOTE routine sets the REN bus line.

First IEEE routine The first IEEE routine to execute after MINC is started invokes IEEE_BUS_CLEAR automatically, which clears and then sets the REN line.

IEEE_BUS_CLEAR The IEEE_BUS_CLEAR routine clears then sets the REN bus line.

TEST_REMOTE

Restrictions	None
Errors	?MINC-F-Invalid argument
Example	None

TEST_SRQ

Test the Service Request (SRQ) Bus Line

Operation

This routine tests the SRQ bus line to determine whether or not any instrument on the bus is requesting service. (See "Status," page 9.)

Statement Form

TEST_SRQ(condition)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
condition	numeric variable name	0; -1	required argument

Example TEST_SRQ(C)
Result C is 0 if the SRQ line is clear, -1 if it is not.

Example TEST_SRQ(S%)
Result TEST_SRQ assigns the value 0 to S% if no instrument is requesting service, and the value -1 if one or more instruments is.

Argument Descriptions

condition The state of the SRQ bus line. TEST_SRQ assigns a value to the condition argument.

Values	*Meaning*
0	The SRQ line is clear.
-1	The SRQ line is set.
Default:	required argument

TEST_SRQ tests the state of the SRQ bus line. Instruments on the bus can request service from MINC by setting this line.

Related Routines

SERIAL_POLL The SERIAL_POLL routine conducts a serial poll to determine which instrument is requesting service. When an instrument that is requesting service is serially polled, it stops requesting service.

SRQ_SUBROUTINE The SRQ_SUBROUTINE routine is used to designate a service subroutine which will be entered automatically whenever an instrument requests service. TEST_SRQ is not needed if service requests are detected this way.

Restrictions None

Errors ?MINC-F-Invalid argument

Example None

TRANSFER

Supervise Message Transfer Between Instruments

Operation

One instrument on the bus, the talker, sends a message string to one or more other instruments, the listeners. MINC also listens so that it can tell when the message is over, but it does not store the message. (See "Transferring Messages," page 9.)

This routine tells the specified instrument to talk by sending the appropriate "my talk address" and "my secondary address" commands. It then tells the specified instruments to listen by sending the "unlisten" command and the appropriate "my listen address" and "my secondary address" commands. MINC then clears the "attention" (ATN) bus line, allowing the talker to send message characters. (See "Commands: the ATN Line," page 20, "Command Codes," page 22, and "Message Codes," page 22.)

Statement Form

TRANSFER(actual-length,maximum-length,talker,listeners)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
actual-length	numeric variable name	0 to 32,767	no value assigned
maximum-length	numeric expression	0 to 32,767	no limit
talker	numeric expressions	0 to 30; 200 to 230	required argument
listeners	numeric expressions	0 to 30; 200 to 230	required argument

Example TRANSFER(,,5,17)
Result Instrument 5 sends a message string of any number of characters to instrument 17.

Example TRANSFER(L,,3,15,2)
Result Instrument 3 sends a message string of any number of characters to instruments 15 and 2. TRANSFER assigns the actual number of characters in the message to the variable L.

Example TRANSFER(N%,20,17,21,213)
Result Instrument 17 sends a message of up to 20 characters to secondary address 13 of instrument 21. TRANSFER assigns the actual number of characters in the message to the variable N%.

Argument Descriptions

actual-length The actual number of characters in the message sent by the talker. TRANSFER assigns a value to the actual-length argument.

Values: 0 to 32,767

Default: No value assigned.

maximum-length The maximum number of characters MINC allows the talker to send.

Values: 0 to 32,767

Default: MINC imposes no limit on the length of the talker's message.

MINC stops the talker from sending any more characters when the number of characters in this message reaches the limit specified in the maximum-length argument. MINC ends the message before this if the talker either sets the END bus line while sending a character or sends a terminator. Terminators are not included in the length of the message reported in the actual-length argument.

talker The address of the instrument that sends the message.

Values: 0 to 30; 200 to 230

Default: required argument

The talker argument is an instrument address (0 to 30), which can be followed by a secondary address (200 to 230). If a secondary address is specified, separate it from the instrument address with a comma.

listeners The addresses of the instruments that receive the message.

Values: 0 to 30; 200 to 230

Default: required argument

The listeners argument is a list of instrument addresses (0 to 30). Each address may be followed by a secondary address (200 to 230). Separate addresses with commas.

Each instrument that can talk or listen has a vocabulary of characters that are meaningful to it. This vocabulary is instrument specific and not defined by the IEEE standard. The message sent by the talker must make sense in terms of the listener's vocabulary. If it does not, the listener might ignore the message or do something unexpected.

Related Routines

RECEIVE The RECEIVE routine receives a string from an instrument. The instrument talks and MINC listens; other instruments can also listen. RECEIVE differs from TRANSFER in that MINC stores the message.

SEND, SEND_FRAGMENT The SEND and SEND_FRAGMENT routines send a string to an instrument. MINC talks and the instrument listens.

SET_TERMINATORS After MINC is started, carriage return and line feed are terminators, characters MINC recognizes as end of message indicators. Some instruments, however, use other terminators. Use the SET_TERMINATORS routine to specify terminators.

Restrictions

None

Errors

?MINC-F-Invalid argument

?MINC-F-Invalid instrument address

?MINC-W-Same instrument specified as talker and listener

MINC prints this warning but allows the message to be transmitted.

?MINC-F-Instrument time limit exceeded

The talker specified is either not on the bus, turned off, not designed to talk, set locally to "listen only," or not at the address specified. Some instruments need a longer time limit, which you can specify with the INSTR_TIME_LIMIT routine.

?MINC-F-Listener is not on the bus

No instrument is on the bus

Example

Assume that instrument 1 is a line printer and instrument 2 is a voltmeter. The following program transfers a message from the voltmeter to the line printer.

```
10 IEEE_BUS_CLEAR \ ALL_INSTR_CLEAR
20 P=1 \ V=2
30 TRANSFER(,,V,P)
40 STOP
```

Statement 10 clears the bus and all instruments on it. When statement 30 executes, the voltmeter talks and the line printer listens, printing each character sent by the voltmeter.

TRIGGER_INSTR

Trigger Selected Instruments

Operation

Each instrument specified starts its basic operation. The response to this trigger is instrument specific. (See "Triggering," page 9.)

This routine tells the specified instruments to listen by sending the "unlisten" command and the appropriate "my listen address" and "my secondary address" commands. It then sends the "group execute trigger" addressed command. (See "Commands: the ATN Line," page 20, and "Command Codes," page 22.)

Statement Form

TRIGGER_INSTR(listeners)

Argument	*Type of Argument*	*Valid Values*	*Default Value*
listeners	numeric expressions	0 to 30; 200 to 230	required argument

Example TRIGGER_INSTR(4)
Result Instrument 4 starts its basic operation.

Example TRIGGER_INSTR(7,15,3)
Result Instruments 7, 15, and 3 each start their basic operation.

Argument Descriptions

listeners The addresses of the instruments to be triggered.

Values: 0 to 30; 200 to 230

Default: required argument

The listeners argument is a list of instrument addresses (0 to 30). Each address can be followed by a secondary address (200 to 230), though secondary addresses are not usually meaningful in this routine. Separate addresses with commas.

Related Routines

RECEIVE The TRIGGER_INSTR routine is sometimes used in conjunction with the RECEIVE routine. TRIGGER_INSTR might tell an instrument to take a reading; the results of that reading are reported in the next RECEIVE routine.

SEND Some instruments ignore triggers unless they have been sent a message telling them to expect a trigger.

Restrictions

Some instruments do not respond to triggers. If an instrument ignores a trigger, no error is produced, and other bus instruments are still triggered.

Errors

?MINC-F-Invalid instrument address

?MINC-F-Listener is not on the bus

The instrument you specified as a listener is either not on the bus, turned off, not designed to listen, set locally to "talk only," or not at the address specified.

Example

See also the examples for the SRQ_SUBROUTINE and PAR_POLL routines.

Your experiment involves a relay box, two voltmeters, and an ohmmeter. The following program closes the relay and then takes simultaneous voltage and resistance readings.

```
10 IEEE_BUS_CLEAR \ ALL_INSTR_CLEAR
20 R=7 \ V1=3 \ V2=12 \ O=6
30 TRIGGER_INSTR(R)
40 TRIGGER_INSTR(V1,V2,O)
50 RECEIVE(V1$,,V1) \ PRINT "Voltage 1 =";V1$
60 RECEIVE(V2$,,V2) \ PRINT "Voltage 2 =";V2$
70 RECEIVE(O$,,O) \ PRINT "Resistance =";O$
80 STOP
```

Statement 10 clears the bus and all the instruments on it. Statement 20 contains the address of the relay box, the two voltmeters, and the ohmmeter.

The first TRIGGER_INSTR statement closes the relay box. The second one causes the two voltmeters and the ohmmeter to take readings simultaneously. In statement 50, one voltmeter sends the reading it took when it was triggered. In statement 60, the other voltmeter sends the reading it took when it was triggered. In statement 70, the ohmmeter sends the reading it took when it was triggered.

INDEX

IEEE Bus Addresses

Instrument	Address	Secondary Addresses

MR-2143

MINC IEEE
Bus Programming
AA-D801A-TC

READER'S COMMENTS

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

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

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

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

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

Name_____ Date_____

Organization_____

Street_____ Telephone_____

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/E37
MARLBOROUGH, MASSACHUSETTS 01752

Do Not Tear - Fold Here and Tape

Cut Along Dotted Line