GPIB11 VMS
Software Reference Manual

October 1993 Edition

Part Number 320300-01

National Instruments Corporate Headquarters
6504 Bridge Point Parkway
Austin, TX 78730-5039
(512) 794-0100
Technical support fax: (800) 328-2203
(512) 794-5678

Branch Offices:
Australia (03) 879 9422, Austria (0662) 435986, Belgium 02/757.00.20, Canada (Ontario) (519) 622-9310,
Canada (Québec) (514) 694-8521, Denmark 45 76 26 00, Finland (90) 527 2321, France (1) 48 14 24 24,
Germany 089/741 31 30, Italy 02/48301892, Japan (03) 3788-1921, Netherlands 03480-33466, Norway 32-848400,
Spain (91) 640 0085, Sweden 08-730 49 70, Switzerland 056/27 00 20, U.K. 0635 523545

Limited Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

Product and company names listed are trademarks or trade names of their respective companies.

Warning Regarding Medical and Clinical Use of National Instruments Products

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.

Contents

Figures

Tables

About This Manual

The *GPIB11 VMS Software Reference Manual* describes the National Instruments GPIB11 software for the VMS operating system. This manual contains instructions for configuring and installing the GPIB11 software, and describes how to use the supplied high-level and low-level driver routines to communicate with an IEEE 488 device. The material in this manual is for users who are familiar with programming on a VMS-based machine. Knowledge of the VMS operating system and the IEEE 488 Standard Digital Interface for Programmable Instrumentation (GPIB) is also helpful.

Organization of This Manual

This manual is organized as follows:

- Chapter 1, *Introduction,* contains a brief description of the software package.
- Chapter 2, *Software Installation and Configuration,* contains instructions for installing and configuring the GPIB11 Series Multiboard Driver software in a VMS or MicroVMS operating system.
- Chapter 3, *IBUP High-Level Routines,* contains information for using the high-level IBUP functions. This chapter also contains a description of each IBUP function. The descriptions are listed alphabetically for easy reference.
- Chapter 4, *GPIB Low-Level Routines,* contains information for using the low-level GPIB functions. This chapter also contains a description of each GPIB function. The descriptions are listed alphabetically for easy reference.
- Chapter 5, *IMICPX,* introduces you to the Interface Bus Interactive Control (`IMICPX`) program. This chapter also contains instructions for running `IMICPX` and lists the conventions for using `IMICPX`.
- Appendix A, *Multiline Interface Messages,* lists the multiline interface messages and describes the mnemonics and messages that correspond to the interface functions. These functions include initializing the bus, addressing and unaddressing devices, and setting device modes for local or remote programming. The multiline interface messages are IEEE 488-defined commands that are sent and received with ATN TRUE.
- Appendix B, *Error Code Summary,* contains a list of error codes returned from the driver functions.

- Appendix C, *FORTRAN Language Call Syntax*, contains a list of FORTRAN language call syntax for high-level and low-level functions.
- Appendix D, *Verifying Hardware and Software Configuration and Installation*, contains instructions for checking the configuration and installation of your interface and the GPIB11 Series VMS software.
- Appendix E, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.
- The *Glossary* contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.

Conventions Used in This Manual

The following conventions are used in this manual:

italic	Italic text denotes emphasis, a cross reference, or an introduction to a key concept.
`monospace`	Lowercase text in this font denotes text or characters that are to be literally input from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, variables, filenames, and extensions, and for statements and comments taken from program code.
<>	Angle brackets enclose the name of a key on the keyboard–for example, <PageDown>.
-	A hyphen between two or more key names enclosed in angle brackets denotes that you should simultaneously press the named keys–for example, <Control-Alt-Delete>.
<Control>	Key names are capitalized.
IEEE 488	IEEE 488 is used throughout this manual to refer to the ANSI/IEEE Standard 488.1-1987, which defines the GPIB.
`IMICPX`	`IMICPX` refers to the Interface Bus Interactive Control program.

Abbreviations, acronyms, metric prefixes, mnemonics, symbols, and terms are listed in the *Glossary*.

Related Documentation

The following document contains information that you may find helpful as you read this manual:

- *VAX Architecture Handbook.*
- ANSI/IEEE Standard 488.1-1987, *ANSI/IEEE Standard Digital Interface for Programmable Instrumentation.*

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. These forms are in Appendix E, *Customer Communication*, at the end of this manual.

Chapter 1
Introduction

This chapter contains a brief description of the software package.

The VMS software package delivered with National Instruments GPIB-series interfaces consists of two modules: a high-level utility subroutine and a low-level utility subroutine. Figure 1-1 shows these two modules.

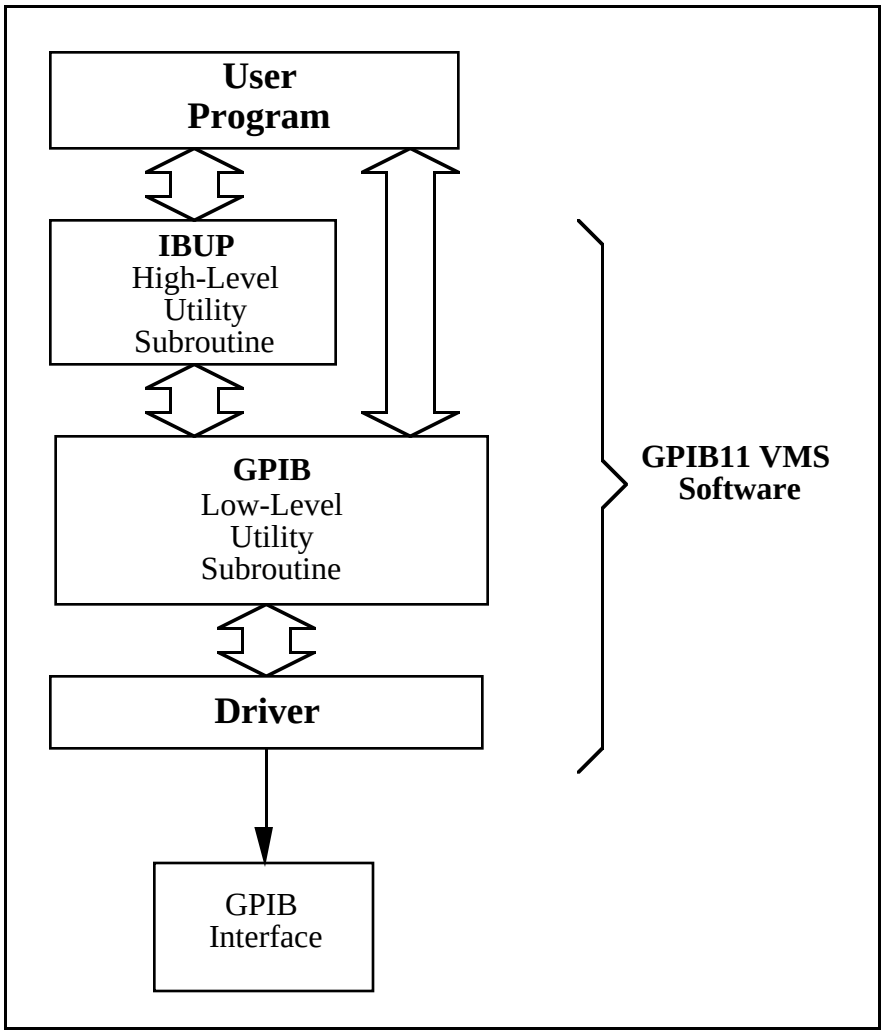

Figure 1-1. GPIB11 VMS Software

The first module is the high-level utility subroutine. This subroutine contains information about the instruments connected to the IEEE Standard 488.1-1987 Standard Digital Interface for Programmable Instrumentation, known as the General Purpose Interface Bus (GPIB). The high-level utility subroutine, known as the Interface Bus Utility Program (IBUP), uses this information to relieve the application programmer of some of the addressing protocol inherent in many GPIB operations. IBUP uses the second module, the low-level utility subroutine (named GPIB), to conduct the actual bus I/O. A user program normally calls the IBUP subroutine; however, in those circumstances where additional flexibility is required, the GPIB subroutine can be called directly.

An interactive control program, known as the Interface Bus Interactive Control (`IMICPX`) program, is included in the standard software package. Using `IMICPX`, you can call any of the utility subroutine functions directly from the computer keyboard. With this feature, you can check the installation of the hardware and the software. The control program aids in the development of application software through single-step control and operation of the GPIB devices on the bus.

The software is designed specifically for the VMS operating system. The utility subroutines are callable from FORTRAN and MACRO application programs and are supplied in MACRO Assembler source form.

Terms Used in This Manual

In this manual, the term *bus* always refers to the General Purpose Interface Bus – as opposed to the computer bus in which an interface is installed. The terms *data* and *data message* are used in place of the IEEE Standard 488 term *device-dependent message*, while the terms *command* and *command message* are used in place of the term *interface message*. The term *interface* refers to the GPIB-series interface that is controlled by the software package. The terms *driver* and *handler* refer to software which is installed as part of the operating system kernel for the purpose of controlling a GPIB interface.

Chapter 2
Software Installation and Configuration

This chapter contains instructions for installing and configuring the GPIB11 Series Multiboard Driver software in a VMS or MicroVMS operating system.

Note: If you have the National Instruments GPIB-SCSI interface installed on the DEC VAXstation 3100 or MicroVAX 3100, refer to the getting started manual that came with your hardware for software installation and configuration information.

Required Distribution Files

Your VAX/VMS multiboard software distribution diskette contains the following files:

`IMDRVX.MAR` is the VAX/VMS multiboard device driver. The name of this file should be changed to `IBDRIVER.MAR` after you copy it from the original distribution medium.

`IMGPIB.MAR` is a file containing the low-level driver interface routines.

`IMIBUP.MAR` is a file containing the high-level utility routines.

`IMICPX.FOR` is the interactive control program for debugging and troubleshooting GPIB systems.

Assembly Parameter Editing

The VAX/VMS multiboard device driver is supplied as a loadable driver. Several parameters in the driver file (`IMDRVX.MAR`) must be edited. Locate the *Editable Parameters* section of the driver file and modify the specified parameters as follows:

- INTERFACE Macros

  Each interface must be defined in a Macro call. The appropriate values are described from left to right as follows:

  – GPIB primary address. Enter the primary address of your GPIB interface board.

  – GPIB secondary address. Enter the secondary address of your GPIB interface board.

  - SAC. Enter a value of 0 if this board will not be required to operate as System Controller, or a value of 1 if the board may be required to operate as System Controller. This value must match the switch setting of the GPIB interface SAC switch.

  - TRI. Enter a value of 1 to drive the bus tri-state for high-speed timing. Enter a value of 0 to drive the bus open collector.

- uVAX

  Set this value to 0 if you are installing this software on a VAX-11 series computer (using a GPIB11-2), 1 for a MicroVAX I (using a GPIB11v-2), or 2 for a MicroVAX II or MicroVAX 3xxx (using a GPIB11v-2).

- PASSCT

  If the Pass Control feature of the bus will not be used by any board controlled by this driver, a value of 0 for this constant will reduce the size of the driver code. The code is retained if a non-zero value is used.

- PPENAB

  If the Parallel Poll feature of the bus will not be used by any board controlled by the driver, a value of 0 may be supplied for this constant. The resulting driver will require less memory.

- DIAG

  Set this constant to 1 if diagnostic tracing is desired, or 0 to omit the code from the driver.

Note: The `IMIBUP.MAR` file requires editing to construct a bus table. Chapter 3, *IBUP High-Level Routines,* contains an example of a bus table.

VAX/VMS Driver Installation

After editing all parameters, complete the following installation procedures to install the driver.

1. Assemble the driver.

```
$ MACRO IBDRIVER                                    <Enter>
```

2. Create a file containing the link options.

```
$ CREATE IBDRIVER.OPT                               <Enter>
BASE=0                                              <Enter>
<Ctrl> <z>                                          <Enter>
```

3. Link the driver.

```
$ LINK/NOTRACE IBDRIVER,IBDRIVER/OPT,
SYS$SYSTEM:SYS.STB/SELECT                           <Enter>
```

The linker will report that the image has no transfer address.

4. Copy IBDRIVER.EXE to the SYS$SYSTEM directory. If the driver has previously been installed, the computer must be restarted to install the new version.

```
$ COPY IBDRIVER.EXE SYS$SYSTEM:*.*                    <Enter>
```

5. Run the SYSGEN utility from the system manager's account to connect all desired boards (board names are of the form IBA0, IBB0, IBC0, and so on.)

   Note: Because the device Control-Status Register is the third of the four device registers, the CSR address specified in the CONNECT command should be 4 plus the value set in the address switches on the board itself.

   The following example shows three GPIB interface boards being installed.

```
$ RUN SYS$SYSTEM:SYSGEN
SYSGEN>CONNECT IBA0 /ADAPTER=3 /CSR=%O767714 /VECTOR=%O310
SYSGEN>CONNECT IBB0 /ADAPTER=3 /CSR=%O767724 /VECTOR=%O320
SYSGEN>CONNECT IBC0 /ADAPTER=3 /CSR=%O767734 /VECTOR=%O330
```

   where ADAPTER (Nexus), CSR, and VECTOR values are system dependent.

   Note: In the preceding command, do not confuse the letter O in %O with the number zero (0).

   To have the boards configured during the startup sequence, SYSGEN should be invoked from a startup command file (for example, SYCONFIG.COM) to connect the boards with the proper ADAPTER, CSR, and VECTOR values.

   The following installation example shows the typical sequence of commands you must follow to install the Multiboard Driver into a VMS operating system.

VAX Installation Example

The following example is the typical sequence of commands necessary to install the driver and run the interactive communication program IMICPX on a VAX/MicroVAX computer.

Note: Some of the files must be edited before they are assembled and the values used in the SYSGEN CONNECT command are system dependent.

Entered or Displayed at Terminal	Explanation
`$ SET/PROC/PRIV=CMKRNL` `$ SET/PROC/PRIV=CMEXEC`	Set priorities to allow loading of a driver.
`$ MOUNT DUA1: GPIB`	Initialize drive.
`$ COPY DUA1:[GPIB]*.**.*`	Copy files to GPIB directory.
`$ RENAME IBDRVX.MAR IBDRIVER.MAR` `$ MAC IBDRIVER.MAR`	Rename the driver file. Assemble the driver (Check editable parameters).

(continues)

Entered or Displayed at Terminal	Explanation

`$ CREATE IBDRIVER.OPT`
`BASE=0`
`<Ctrl> <z>`
	Create an option file.

`$ LINK/NOTRACE IBDRIVER,IBDRIVER/OPT, SYS$SYSTEM:SYS.STB/SEL` `%LINK-W-USRTFR, Image "IBDRIVER" has no user transfer address`	Link the driver.
`$ SET PROC/PRIV=BYPASS` `$ COPY IBDRIVER.EXE SYS$SYSTEM:*.*`	Copy to driver directory.
`$ RUN SYS$SYSTEM:SYSGEN`	

`SYSGEN>  CONNECT IBA0 /ADAPTER=0 /CSR=%O767714 /VECTOR=%O310`
`SYSGEN>  CONNECT IBB0 /ADAPTER=0 /CSR=%O767724 /VECTOR=%O320`
`SYSGEN>  EXIT`
	`SYSGEN` GPIB boards.

`$ MAC IMIBUP` `$ MAC IMGPIB`	Assemble library files (Check bus table in `IMIBUP`).
`$SET PROC/PRIV=SYSLCK`	Allow lock service for IBUP.
`$ FORTRAN /EXTEND_SOURCE IMICPX` `$ LINK IMICPX+IMIBUP+IMGPIB`	Compile `IMICPX` and link to library files.
`$ RUN IMICPX`	Run `IMICPX` to begin programming GPIB.

Chapter 3
IBUP High-Level Routines

This chapter contains information for using the high-level IBUP functions. This chapter also contains a description of each IBUP function. The descriptions are listed alphabetically for easy reference.

The IEEE 488 Bus Utility program (IBUP) performs convenient high-level programming of the GPIB by automatically handling the complexities of bus addressing and protocol and presenting a simple, understandable user interface. An IBUP routine makes one or more calls to a low-level utility routine (a GPIB routine) to perform its functions. Internal to the IBUP routine, a GPIB Device Table contains addressing and data transfer information specific to each instrument connected to the bus. Thus, the target instrument for any IBUP operation is referred to by its device table slot number, analogous to a *logical unit number*. Some specialized functions that are not provided by the IBUP utility routine can be performed by calling the GPIB routine directly.

All IBUP calls are synchronous, meaning that the return to the calling program is not done until the I/O completes successfully or terminates on an error. Also, with the exception of the `DEFINE` and `FINISH` functions, all IBUP functions return an error if the call is made while the interface is *not* the System Controller and/or Controller-In-Charge.

GPIB Device Table

The GPIB Device Table identifies each GPIB interface that is connected to the computer and describes all the devices that are connected to each bus. The table is organized by bus groups. Each group is headed by the `BUS` macro, numbered sequentially beginning with `BUS 0`, and followed by a set of device description macros. These `DEVICE` macros are implicitly numbered sequentially within each group, starting with 0. `DEVICE 0` defines the GPIB interface that is used to execute `IBUP[U]` (where `[U]` signifies a unit number in the syntax) functions on a bus. This interface also has an associated unit number that is specified in the driver. Thus, `DEVICE 0` of `BUS 2` is also unit 2 in the driver. Other than this restriction for `DEVICE 0`, the other `DEVICE` macros can be entered in any order.

Each entry in the device table contains six bytes which indicate the GPIB talk address (`TTT`), GPIB listen address (`LLL`), GPIB secondary address (`SSS`), read mode (`R`), write mode (`W`), and end-of-data character (`EOD`) of the instrument. The instrument is then referred to by its location, or device number, in the device table for all calls to the IBUP routines. (device `0` is reserved for the GPIB interface.) The size of the device table is a *compile-time* parameter. As many devices as are needed can be statically configured directly in the source code, or a number of unused devices can be included, which then can be dynamically configured at run-time (see the *DEFINE* function described later in this chapter).

Referring to the IMIBUP.MAR source file, a DEVICE in the device table is a line in the data structure containing the required information, as shown in the following example:

```
DEVICE  TTT, LLL, SSS, R, EOD, W
```

TTT is the octal talk address (MTA or 0 if the device cannot be a Talker). LLL is the octal listen address (MLA or 0 if the device cannot be a Listener). SSS is the octal secondary address (MSA or 0 if extended addressing is not being used). R is the data transfer mode to use when reading from the device. EOD is the end-of-data character (used with certain read modes). W is the data transfer mode to use when writing to the device. Refer to Tables 4-1 and 4-2 for descriptions of the read and write data transfer modes, respectively, and to Appendix A, *Multiline Interface Messages*, for the appropriate talk and listen addresses.

Each device on the bus is assigned a unique 5-bit GPIB primary address, which in turn determines the talk and listen addresses for that device. In addition, some devices may respond to one or more secondary addresses. The method of setting the GPIB primary address for each device is found in the instruction manual for that device. The talk address (TTT) resulting from the bus address setting is determined by adding octal 100 to the octal value of the device address setting. Adding octal 040 to the octal value of the device address setting determines the listen address (LLL). Adding octal 140 to the 5-bit secondary device address setting computes the secondary address (SSS), if there is one. In the device table, record the talk, listen, and secondary addresses for all devices on the bus.

Note: Never set any bus address to all ones; the resulting talk and listen addresses for that device would conflict with the commands Untalk (UNT) and Unlisten (UNL).

Example GPIB Device Table

The following example shows a bus table for a system with four buses.

```
BUS  0
DEVICE  <^O100>, <^O40>, <^O000>, 0,  0,  0  ;  GPIB interface 0
DEVICE  <^O101>, <^O41>, <^O000>, 0,  0,  2  ;  Device 1
DEVICE  <^O102>, <^O42>, <^O142>, 0,  0,  2  ;  Device 2
DEVICE  <^O103>, <^O43>, <^O143>, 0,  0,  2  ;  Device 3
BUS  1
DEVICE  <^O100>, <^O40>, <^O000>, 0,  0,  0  ;  GPIB interface 1
DEVICE  <^O105>, <^O45>, <^O000>, 0,  0,  2  ;  Device 1
BUS  2
DEVICE  <^O111>, <^O51>, <^O000>, 0,  0,  2  ;  GPIB interface 2
DEVICE  <^O105>, <^O45>, <^O000>, 0,  0,  2  ;  Device 1
BUS  3
DEVICE  <^O100>, <^O40>, <^O000>, 0,  0,  2  ;  GPIB interface 3
DEVICE  <^O106>, <^O46>, <^O000>, 0,  0,  2  ;  Device 1
DEVICE  <^O107>, <^O47>, <^O000>, 0,  0,  2  ;  Device 2
DEVICE  <^O110>, <^O50>, <^O000>, 0,  0,  2  ;  Device 3
DEVICE  <^O111>, <^O51>, <^O000>, 0,  0,  2  ;  Device 4
```

Program Calling Syntax

The IBUPU calling syntax (where the second U identifies the syntax as containing a unit number) contains the following parameters.

- unit number
- function code
- zero or more arguments for the function

For compatibility with earlier versions of the utility software that accepted single-board control only, the IBUP calling syntax is preserved. With this syntax, the unit number is assumed to be 0 and the UNIT argument is omitted from the parameter list. Consequently, an IBUPU call in FORTRAN written as follows:

```
J=IBUPU(0,FUNCTN,ARG1,...,ARGn)
```

can be replaced with an IBUP call written as follows:

```
J=IBUP(FUNCTN,ARG1,...,ARGn)
```

FORTRAN Call

To use IBUP[U], the calling program must declare IBUP and IBUPU as integer-value functions. To perform an IBUP[U] function, the calling program executes a FORTRAN function call as follows:

```
J=IBUPU(UNIT,FUNCTN,ARG1,...,ARGn)
```

or

```
J=IBUP(FUNCTN,ARG1,...,ARGn)
```

where UNIT is the unit number of the interface that is used; FUNCTN is an integer type variable or constant indicating the function code, and ARG1 through ARGn are any argument expressions required by the particular function. The value returned by IBUP or IBUPU indicates the success or failure of the operation. When IBUP is used in a multiboard environment, the unspecified value for the board designation is assumed to be zero. A summary of the syntax for FORTRAN calls can be found in Appendix C, *FORTRAN Language Call Syntax*, in this manual.

Macro Call

Calling utility routine functions from an Assembly language program is done via the VAX/VMS standard procedure calling mechanism, as described in the *VAX Architecture Handbook.* In the CALLS format, the arguments are pushed onto the stack in reverse (right to left) order, and the call is made with the following instructions:

```
CALLS   #N, G^IBUPU
CALLS   #N, G^IBUP
```

where N is the number of arguments passed.

In the CALLG format, the arguments are placed in the standard argument block, and the call is made with the following instructions:

```
CALLG   ARGADDR, G^IBUPU
CALLG   ARGADDR, G^IBUP
```

where ARGADDR is the address of the argument block. All arguments are passed by reference. On return from IBUP[U], register r0 contains the value of the function as defined in the FORTRAN call. In the case of certain errors (especially system-related errors), register r1 may contain additional information concerning the error. All other registers are preserved across the call.

A summary of the syntax for Macro calls can be found in Appendix C, *FORTRAN Language Call Syntax*.

IBUP High-Level Functions

The remainder of this chapter contains a detailed description of each IBUP function with examples. A negative value returned by a function indicates an error. Referenced GPIB functions in Chapter 4, *GPIB Low-Level Routines*, describe the possible error conditions that can be returned. (Refer to Appendix B, *Error Code Summary*, for a complete listing of all error codes.)

CLEAR

Function Code: 2

Format:
```
RESULT=IBUPU(UNIT,2,D)
          or
RESULT=IBUP(2,D)
```

Remarks: `UNIT` is the interface unit number. `2` is the function code. Integer `D` is the device number. The value returned in `RESULT` is 1 unless an error occurred, in which case the return value is the code returned by the GPIB `COMMAND` function or GPIB `CLEAR` function.

The `CLEAR` function selectively clears one device on the GPIB, simultaneously clears all devices on the GPIB, or initializes the GPIB itself by sending the Interface Clear (IFC) message. If the device number is 0, all devices are cleared (via the DCL interface message). If the device number is negative, the GPIB interface is initialized (via the IFC uniline message). Otherwise, only the selected device is cleared (via the SDC interface message).

The `CLEAR` function calls the GPIB `COMMAND` function to perform addressing and to send the Selected Device Clear (SDC) or Device Clear (DCL) message. The GPIB `CLEAR` function is called to send the IFC message. For more details, refer to the GPIB routine descriptions in Chapter 4, *GPIB Low-Level Routines*.

Examples:

```
C Send the Interface Clear (IFC) command.
C
    RESULT = IBUP(2, -1)

C
C Clear all devices by sending the Device Clear
C (DCL) command.
C
    RESULT = IBUP(2, 0)

C
C Clear Device 3 by sending its listen address
and
C the Selected Device Clear (SDC) command.
C
    RESULT = IBUP(2, 3)
```

CONFIGURE

Function Code: 7

Format:
```
RESULT=IBUPU(UNIT,7,D,S,L)
          or
RESULT=IBUP(7,D,S,L)
```

Remarks: UNIT is the interface unit number. 7 is the function code. Integer D is the device number. Integer S is the sense of the parallel poll response. L is an integer representing the data line for the response. The value returned in RESULT is 1 unless an error occurred, in which case the return value is the code returned by the GPIB COMMAND function.

The CONFIGURE function configures a selected device for parallel polling or unconfigures all devices. If the device number is 0, all devices are unconfigured. Otherwise, the selected device is configured to respond positively on line L when S is greater than 0 and the device's individual status bit (ist) is true, or when S equals 0 and the status bit is false. The response lines are numbered 1 to 8, corresponding to bits 0 to 7 in the parallel poll byte.

The CONFIGURE function calls the GPIB COMMAND function to perform addressing and to send the Parallel Poll Enable (PPE), Parallel Poll Disable (PPD), Parallel Poll Configure (PPC), and Parallel Poll Unconfigure (PPU) messages. For more details, refer to the GPIB routine descriptions in Chapter 4, *GPIB Low-Level Routines.*

Examples:

```
C Configure Device 3 to respond positively on
DIO4
C during parallel polls when ist = 1.
C
    RESULT = IBUP(7, 3, 1, 4)

C
C Configure Device 4 to respond positively on
DIO6
C during parallel polls when ist = 0.
C
    RESULT = IBUP(7, 4, 0, 6)

C
C Unconfigure all devices.
C
    RESULT = IBUP(7, 0)
```

DEFINE

Function Code: 9

Format:
```
RESULT=IBUPU(UNIT,9,D,TAD,LAD,SAD,RMD,EOD,WMD)
```
or
```
RESULT=IBUP(9,D,TAD,LAD,SAD,RMD,EOD,WMD)
```

Remarks: `UNIT` is the interface number. `9` is the function code. Integer `D` is the device number. Integer `TAD` is the GPIB talk address (0100 to 0136 octal, or 0 if the device is not a Talker). Integer `LAD` is the GPIB listen address (040 to 076 octal, or 0 if the device is not a Listener). `SAD` is an integer representing the GPIB secondary address (0140 to 0176 octal, or 0 if the device does not have a secondary address). `RMD` and `WMD` are integers representing the read and write modes, respectively (see Tables 4-1 and 4-2 in Chapter 4, *GPIB Low-Level Routines*). Integer `EOD` is the end-of-data character for certain read modes. The value returned in `RESULT` is a 1 to indicate success or a negative number to indicate an error.

The `DEFINE` function replaces the information in the device table with the arguments defined in the call. No GPIB operation is performed.

Note: You can execute this IBUP function without an error when the interface is not the System Controller or Controller-In-Charge.

Example:

```
C Define Device 3 to be at talk address 0103
octal
C and listen address 043 octal.  Device 3 has no
C secondary address and uses Read Mode 0
C (terminate on count or EOI) and
C Write Mode 2 (send EOI).
C
    RESULT = IBUP(9,3,'103'O,'43'O,0,0,0,2)
```

FINISH

Function Code: 10

Format:
```
RESULT=IBUPU(UNIT,10)
          or
RESULT=IBUP(10)
```

Remarks: `UNIT` is the interface number. `10` is the function code. The `FINISH` function terminates usage of the GPIB by calling the GPIB `FINISH` function to disable interrupts, reset the interface, and close the I/O channel if necessary. The value returned in `RESULT` is a 1 to indicate success, or a negative number to indicate an error.

Note: You can execute this IBUP function without an error when the interface is not the System Controller or Controller-In-Charge.

Example:
```
C Reset the GPIB hardware and software.
C
      RESULT = IBUP(10)
```

LOCAL

Function Code: 5

Format:
```
RESULT=IBUPU(UNIT,5,D)
          or
RESULT=IBUP(5,D)
```

Remarks: `UNIT` is the interface number. `5` is the function code. Integer `D` is the device number. The value returned in `RESULT` is 1 unless an error occurred, in which case the return value is the code returned by the GPIB `COMMAND` function or GPIB `LOCAL` function.

The `LOCAL` function returns a *selected* device to Local mode, returns *all* devices to Local mode, or returns *all* devices to Local mode and *cancels* the Local Lockout condition. If the device number is 0, all devices are returned to Local mode. If the device number is negative, all devices are returned to Local mode and the Local Lockout condition is canceled. Otherwise, only the selected device is returned to Local mode.

The `LOCAL` function calls the GPIB `COMMAND` function to perform addressing and to send the Go To Local (GTL) message. The GPIB `LOCAL` function is called to cancel the REN and LLO messages. For more details, refer to the GPIB routine descriptions in Chapter 4, *GPIB Low-Level Routines*.

Examples:

```
C Return Device 3 to local mode by sending its
C listen address and the Go To Local (GTL)
C command.
C
    RESULT = IBUP(5, 3)

C
C Return all devices to local mode by sending
C their listen addresses and the Go To Local
C command.
C
    RESULT = IBUP(5, 0)

C
C Return all devices to local mode and cancel the
C Local Lockout condition by unasserting the
C Remote Enable (REN) line.
C
    RESULT = IBUP(5, -1)
```

(GTL)

PASS CONTROL

Function Code: 8

Format:
```
RESULT=IBUPU(UNIT,8,D)
          or
RESULT=IBUP(8,D)
```

Remarks: UNIT is the interface number. 8 is the function code. Integer D is the device number. The value returned in RESULT is 1 unless an error occurred, in which case the return value is the code returned by the GPIB COMMAND function or the GPIB PASS CONTROL function.

The PASS CONTROL function allows the selected device (with Controller capabilities) to control the GPIB.

The PASS CONTROL function calls the GPIB COMMAND function to perform addressing and to send the Take Control (TCT) message. The GPIB PASS CONTROL function is called to allow the selected device to become the Controller-In-Charge. For more details, refer to the GPIB routine descriptions in Chapter 4, *GPIB Low-Level Routines.*

Example:
```
C Pass control to Device 3 by sending its talk
C address and the Take Control (TCT) command.
C
    RESULT = IBUP(8, 3)
```

POLL

Function Code: 6

Format:

```
RESULT=IBUPU(UNIT,6,D)
          or
RESULT=IBUP(6,D)
```

Remarks: UNIT is the interface number. 6 is the function code. Integer D is the device number. The value returned in RESULT is the status byte (or parallel poll byte) unless an error occurred, in which case the return value is the code returned by the GPIB COMMAND, GPIB READ, GPIB TESTSRQ, or GPIB PARALLEL POLL function.

The POLL function serially polls a selected device, parallel polls all devices, or tests if any device is requesting service. If the device number is 0, all devices are polled in parallel. If it is negative, the status of the SRQ line is returned (which is negative 1 if no device is requesting service and 1 if at least one device is requesting service and the GPIB interface is the Controller-In-Charge; that is, able to service the request). Otherwise, the selected device is polled serially and its status byte is returned.

The POLL function calls the GPIB COMMAND and GPIB READ functions to conduct a serial poll, the GPIB PARALLEL POLL function to conduct a parallel poll, and the GPIB TESTSRQ function to determine whether a device is requesting service. Notice that the parallel poll byte is logically ORed with 0400 octal before being returned. For more details, refer to the GPIB routine descriptions in Chapter 4, *GPIB Low-Level Routines*.

Examples:

```
C Test if any device is requesting service via
the
C SRQ line.
C
    RESULT = IBUP(6, -1)

C
C Parallel poll all devices and return the
C response in "RESULT" ORed with 0400 octal.
C
    RESULT = IBUP(6, 0)

C
C Serial poll Device 3 and return the response in
C "RESULT".
C
    RESULT = IBUP(6, 3)
```

READ

Function Code: 1

Format:
```
RESULT=IBUPU(UNIT,1,D,ARRAY,COUNT)
          or
RESULT=IBUP(1,D,ARRAY,COUNT)
```

Remarks: UNIT is the interface number. 1 is the function code. Integer D is the device number. ARRAY is a byte array into which bytes received from the device are placed. Integer COUNT is the number of bytes (0 to 65,535) to read. The value returned in RESULT is the number of bytes read for a successful read, or the code returned by the GPIB COMMAND function or GPIB READ function if an error occurs.

The READ function addresses the selected device as Talker and transfers the data bytes from that device to the computer using the read mode defined in the GPIB Device Table (see Table 4-1 in Chapter 4 for the read modes). All devices are then unaddressed following completion of the transfer.

The READ function calls the GPIB COMMAND function to perform the addressing and unaddressing of the GPIB interface and the target device. The READ function calls the GPIB READ function to input the data. For more details, refer to the GPIB routine descriptions in Chapter 4, *GPIB Low-Level Routines*.

Example:
```
BYTE BUFFER(16)

 C
 C Read up to 16 bytes of data from Device 3 into
 C "buffer" and return the actual number of bytes
 C read in "RESULT".
 C
     RESULT = IBUP(1, 3, BUFFER, 16)
```

REMOTE

Function Code: 4

Format:
```
RESULT=IBUPU(UNIT,4,D)
          or
RESULT=IBUP(4,D)
```

Remarks: `UNIT` is the interface number. `4` is the function code. Integer `D` is the device number. The value returned in `RESULT` is 1 unless an error occurred, in which case the return value is the code returned by the GPIB `COMMAND` function or GPIB `REMOTE` function.

The `REMOTE` function places *a* selected device in remote mode, places *all* devices in remote mode, or places *all* devices in remote mode *with* Local Lockout. If the device number is 0, all devices are placed in remote mode. If it is negative, all devices are placed in remote mode with Local Lockout. Otherwise, only the selected device is placed in remote mode. Unless the Local Lockout mode is used, an interfaced device can override the remote program control by means of a front panel Local mode switch.

The `REMOTE` function calls the GPIB `COMMAND` function to perform addressing and to send the Local Lockout (LLO) message. The GPIB `REMOTE` function is called to send the Remote Enable (REN) message. For more details, refer to the GPIB routine descriptions in Chapter 4, *GPIB Low-Level Routines*.

Examples:

```
C
C Place Device 3 in Remote Mode by asserting the
C Remote Enable (REN) line and sending its listen
C address.
C
    RESULT = IBUP(4, 3)

C
C Place all devices in Remote Mode by asserting
C the REN line and sending all of their listen
C Addresses.
C
    RESULT = IBUP(4, 0)

C
C Place all devices in Remote Mode with Local
C Lockout by asserting the REN line and sending
C the Local Lock Out (LLO) command and all of
C their listen Addresses.
C
    RESULT = IBUP(4, -1)
```

TRIGGER

Function Code: 3

Format:
```
RESULT=IBUPU(UNIT,3,D)
          or
RESULT=IBUP(3,D)
```

Remarks: `UNIT` is the interface number. `3` is the function code. Integer `D` is the device number. The value returned in `RESULT` is 1 unless an error occurred, in which case the return value is the code returned by the GPIB `COMMAND` function.

The `TRIGGER` function selectively triggers one device or simultaneously triggers all devices. If the device number is 0, all devices are triggered; otherwise, only the selected device is triggered.

The `TRIGGER` function calls the GPIB `COMMAND` function to perform addressing and to send the Group Execute Trigger (GET) message. For more details, refer to the GPIB routine descriptions in Chapter 4, *GPIB Low-Level Routines*.

Examples:

```
C
C Trigger Device 3 by sending its listen address
C and the Group Execute Trigger (GET) command.
C
    RESULT = IBUP(3, 3)

C
C Trigger all devices by sending their listen
C Addresses and the GET command.
C
    RESULT = IBUP(3, 0)
```

WRITE

Function Code: 0

Format:
```
RESULT=IBUPU(UNIT,0,D,ARRAY,COUNT)
          or
RESULT=IBUP(0,D,ARRAY,COUNT)
```

Remarks: `UNIT` is the interface number. `0` is the function code. Integer `D` is the device number. `ARRAY` is a string of bytes to write to the device. Integer `COUNT` is the size of the array in bytes (0 to 65,535). The value returned in `RESULT` is the number of bytes transferred if successful, or the code returned by the GPIB `COMMAND` function or GPIB `WRITE` function if an error occurs.

The `WRITE` function addresses the selected device as Listener and transfers the data bytes from the computer to that device using the write mode defined in the GPIB Device Table (see Table 4-2 in Chapter 4, *GPIB Low-Level Routines*, for the write modes). All devices are unaddressed following completion of the transfer.

The `WRITE` function calls the GPIB `COMMAND` function to perform the addressing and unaddressing of the GPIB interface and the target device. The `WRITE` function calls the GPIB `WRITE` function to input the data. For more details, refer to the GPIB routine descriptions in Chapter 4, *GPIB Low-Level Routines*.

Example:
```
C
C Write the data bytes to Device 3.
C
    RESULT = IBUP(0, 3, "F3R7T3", 6)
```

Chapter 4
GPIB Low-Level Routines

This chapter contains information for using the low-level GPIB functions. This chapter also contains a description of each GPIB function. The descriptions are listed alphabetically for easy reference.

The low-level utility routines (the GPIB routines) give you complete flexibility to use all IEEE 488 functions, but require that you be familiar with the interface message protocol described in the ANSI/IEEE Standard 488.1-1987 specification. In most applications, the high-level utility routine has all the functions that are needed. However, in certain circumstances, where additional flexibility is required, the user program can call the GPIB routine directly. Such calls can be freely mixed with calls to high-level routines.

All calls to the utility routines are synchronous, meaning that the return to the calling program is not done until the I/O completes successfully or terminates on an error.

Program Calling Syntax

The `GPIBU` calling syntax (where the second `U` identifies the syntax as containing a unit number) contains the following parameters:

- unit number
- function code
- zero or more arguments for the function

For compatibility with earlier versions of the utility software that accepted single-board control only, the `GPIB` calling syntax is preserved. With this syntax, the unit number is assumed to be `0` and the `UNIT` argument is omitted from the parameter list. Consequently a `GPIBU` call in FORTRAN written as follows:

```
J=GPIBU(0,FUNCTN, ARG1,...,ARGn)
```

can be replaced with a `GPIB` call written as follows:

```
J=GPIB(FUNCTN, ARG1,...ARGn)
```

FORTRAN Call

To use GPIB[U], the calling program must declare GPIB and GPIBU as integer-value functions. To perform a GPIB[U] function, the calling program executes a FORTRAN function call as follows:

```
J=GPIBU(UNIT,FUNCTN,ARG1,...,ARGn)
```

or

```
J=GPIB(FUNCTN,ARG1,...,ARGn)
```

where UNIT is the unit number of the interface that is used, FUNCTN is an integer type variable or constant indicating the function code, and ARG1 through ARGn are any argument expressions required by the particular function. The value returned by GPIB or GPIBU indicates the success or failure of the operation. When GPIB is used in a multiboard environment, the unspecified value for the board designation is assumed to be zero. A summary of the syntax for FORTRAN calls can be found in Appendix C, *FORTRAN Language Call Syntax*, in this manual.

Macro Call

Calling utility routine functions from an assembly language program is done via the VAX/VMS standard procedure calling mechanism, as described in the *VAX Architecture Handbook.* In the CALLS format, the arguments are pushed onto the stack in reverse (right to left) order, and the call is made with the following instructions:

```
CALLS   #N, G^GPIBU
CALLS   #N, G^GPIB
```

where N is the number of arguments passed.

In the CALLG format, the arguments are placed in the standard argument block, and the call is made with the following instructions:

```
CALLG   ARGADDR, G^GPIBU
CALLG   ARGADDR, G^GPIB
```

where ARGADDR is the address of the argument block. All arguments are passed by reference. On return from GPIB[U], register r0 contains the value of the function as defined in the FORTRAN call. In the case of certain errors (especially system-related errors), register r1 may contain additional information concerning the error. All other registers are preserved across the call.

A summary of the syntax for Macro calls can be found in Appendix C, *FORTRAN Language Call Syntax*.

GPIB Low-Level Functions

The remainder of this chapter contains a detailed description of each GPIB function with examples. The descriptions contain information regarding the interaction of the functions with the interface Controller function (specifically, whether or not the interface is required to be the System Controller and/or the Active Controller).

A negative value returned by the function indicates an error. Error codes and descriptions are described in Appendix B, *Error Code Summary*.

CLEAR

Function Code: 4

Format:
```
RESULT=GPIBU(UNIT,4)
          or
RESULT=GPIB(4)
```

Remarks: `UNIT` is the interface number. `4` is the function code. The value returned in `RESULT` is 1 if the operation was successful. A negative number indicates an error.

The `CLEAR` function makes the GPIB interface the System Controller and initializes the bus by sending the IFC message. The function returns an error if the interface is not configured as the System Controller or if the board is unable to become the Active Controller. Refer to Appendix B, *Error Code Summary*, for an interpretation of the various error conditions which can occur.

If the `CLEAR` function is successful, the interface becomes Active Controller as soon as ATN is unasserted by the current Controller-In-Charge.

Example:
```
C
C Send the Interface Clear (IFC) command.
C
     RESULT = GPIB(4)
```

COMMAND

Function Code: 0

Format:
```
RESULT=GPIBU(UNIT,0,ARRAY,COUNT)
          or
RESULT=GPIB(0,ARRAY,COUNT)
```

Remarks: `UNIT` is the interface number. `0` is the function code. `ARRAY` is an array of command bytes to send out on the bus. Integer `COUNT` is the size of the array in bytes (0 to 65,535). If greater than or equal to 0, the value returned in `RESULT` is the number of bytes transferred. A negative returned value indicates an error.

The `COMMAND` function is the only means by which multiline command messages can be sent on the bus. The `IBUP` routine or the user program calls this function to address and unaddress Talkers and Listeners in preparation for data byte transfers (via `READ`, `WRITE`, or `TRANSFER`).

This function is also called to send the following multiline command messages:

DCL - Device Clear	PPE - Parallel Poll Enable
GET - Group Execute Trigger	PPU - Parallel Poll Unconfigure
GTL - Go to Local	SDC - Selected Device Clear
LLO - Local Lockout	SPD - Serial Poll Disable
PPC - Parallel Poll Configure	SPE - Serial Poll Enable
PPD - Parallel Poll Disable	TCT - Take Control

If the interface is the Active Controller, the command messages are output immediately. If it is not Controller-In-Charge, an error is returned.

If the interface is configured as the System Controller, you must call the `CLEAR` function prior to the first call to `COMMAND`.

Example:

```
C
C Send Interface Clear (IFC), unaddress all
C Talkers and Listeners, and then address the
GPIB
C interface at primary address 1 to talk and
C the device at primary address 3 to listen.
C
    RESULT = GPIB(4)
    RESULT = GPIB(0, "?_A#", 4)
```

FINISH

Function Code: 14

Format:
```
RESULT=GPIBU(UNIT,14)
```
or
```
RESULT=GPIB(14)
```

Remarks: `UNIT` is the interface number. `14` is the function code. The value returned in `RESULT` is 1 if the function completes successfully.

The `FINISH` function disables all interface interrupts and resets the interface. The interface remains in a reset state until the next GPIB function call is made.

Example:
```
C
C Reset the GPIB hardware and software.
C
    RESULT = GPIB(14)
```

LOCAL

Function Code: 6

Format:
```
RESULT=GPIBU(UNIT,6)
```
or
```
RESULT=GPIB(6)
```

Remarks: `UNIT` is the interface number. `6` is the function code. The value returned in `RESULT` is 1 if the function is successful. A negative number indicates an error.

The `LOCAL` function places devices in local mode. It returns an error if the interface is not configured as the System Controller. `LOCAL` clears the REN message, forcing all devices on the bus to return to local mode.

Example:

```
C
C Return all devices to local mode and cancel the
C Local Lockout condition by unasserting the
C Remote Enable (REN) line.
C
    RESULT = GPIB(6)
```

PARALLEL POLL

Function Code: 7

Format:
```
RESULT=GPIBU(UNIT,7)
```
or
```
RESULT=GPIB(7)
```

Remarks: `UNIT` is the interface number. `7` is the function code. The value returned in `RESULT` is equal to the result of the `PARALLEL POLL` function, or is a negative number indicating an error.

The `PARALLEL POLL` function polls the bus devices that have been configured for parallel polling. A device can be remotely configured by using the `COMMAND` function or locally configured at the device itself.

If the interface is the Active Controller, the program takes the poll immediately and returns the results. Otherwise, the program returns an error.

The result of the `PARALLEL POLL` function is the 8-bit response which appeared on the bus DIO lines, logically ORed with 0400 octal.

Example:

```
C
C Parallel poll all devices and return the
C response in "RESULT" ORed with 0400 octal.
C
    RESULT = GPIB(7)
```

PASS CONTROL

Function Code: 8

Format:
```
RESULT=GPIBU(UNIT,8)
          or
RESULT=GPIB(8)
```

Remarks: UNIT is the interface number. 8 is the function code. The value returned in RESULT is 1 if the function is successful. A negative number indicates an error.

The PASS CONTROL function is used to allow another GPIB Controller to take charge of the bus. When PASS CONTROL is called, it is assumed that a device with the Controller capability has already been addressed as Talker and has been sent the TCT message with a call to the COMMAND function. The PASS CONTROL function then places the interface Controller function in the idle state, allowing the other Controller to assume active control. If the interface is not the Controller-In-Charge, an error is returned.

When a subsequent WRITE or READ call is made to GPIB and control has not been returned, service is requested from the current Active Controller. The status byte which is presented when the interface is serially polled is octal 102 for WRITE or octal 104 for READ.

If the interface is the System Controller, it can take control from any other Controller at any time by calling the CLEAR function.

Example:

```
C
C Pass control to Device 3 by sending its talk
C address and the Take Control (TCT) command.
C
    RESULT= GPIB(8)
```

READ

Function Code: 2

Format:
```
RESULT=GPIBU(UNIT,2,ARRAY,COUNT,MODE,EOD)
          or
RESULT=GPIB(2,ARRAY,COUNT,MODE,EOD)
```

Remarks: UNIT is the interface number. 2 is the function code. ARRAY is a byte array into which bytes read from the Talker are placed. Integer COUNT is the number of bytes to be read (0 to 65,535). MODE is an integer specifying the read mode (see Table 4-1). Integer EOD is the End-Of-Data character. If greater than or equal to 0, the value returned in RESULT is the actual number of bytes read. A negative returned value indicates an error.

Data bytes are sent from a talking device to the computer only by means of the READ function. It is assumed that another device has been addressed as Talker by using a call to the COMMAND function, or that another Controller is active and has addressed the Talker and all Listeners.

If the interface is the Active Controller when READ is called, the GPIB routine enables the programmable Listen function, puts the Controller on standby, and begins reading data bytes. Otherwise, the subroutine program waits until the interface is addressed as Listener before reading the data. While it is waiting, the subroutine program requests service from the current Active Controller (the status byte, octal 104, reflects that it is requesting to Listen).

Example:

```
     BYTE V(16)
C
C Address the GPIB interface at primary
C address 1 to listen and the device at primary
C address 3 to talk.  Read up to 16 bytes of
C data from the device into V and
C return the actual number of bytes read in
C "RESULT".
C
     RESULT= GPIB(0, "?_C!", 4)

     RESULT= GPIB(2, V, 16, 0, 0)
```

Table 4-1. GPIB READ Modes

Mode	Description
0	Terminate on count or END. Data bytes are read until the requested byte count is reached or the uniline message, END, is detected.
2	Terminate on count or END or EOD. Mode 2 is the same as mode 0 except that reading is also terminated when a byte is read that is equal to the end-of-data (EOD) character.
4	Terminate on count only. Data bytes are read until the requested byte count is reached. END or EOD does not terminate the read.

REMOTE

Function Code: 5

Format:
```
RESULT=GPIBU(UNIT,5)
```
or
```
RESULT=GPIB(5)
```

Remarks: `UNIT` is the interface number. `5` is the function code. The value returned in `RESULT` is 1 if the operation was successful. A negative number indicates an error.

The `REMOTE` function places devices in remote mode. It returns an error if the interface is not configured as the System Controller. `REMOTE` sends the REN message however, devices on the bus are not actually placed in remote mode until they are addressed as Listeners. The REN message remains asserted until the GPIB `LOCAL` function is called.

Example:

```
C
C Send Interface Clear (IFC), assert the Remote
C Enable line, and place the devices at primary
C addresses 1, 3, and 4 in Remote mode.
C
    RESULT = GPIB(4)

    RESULT = GPIB(5)

    RESULT = GPIB(0, "?_!#$", 5)
```

SET PARAMETERS

Function Code: 12

Format:
```
RESULT=GPIBU(UNIT,12,T)
          or
RESULT=GPIB(12,T)
```

Remarks: `UNIT` is the interface number. `12` is the function code. Integer `T` is the time limit in seconds. The value returned in `RESULT` is 1 if the operation was successful, or is a negative number if an error occurred.

The `SET PARAMETERS` function establishes the driver time limit parameter. The time limit is used only for interrupt-driven GPIB functions. The default, or initial, value for the time limit is 10 seconds. If `T` is equal to 0, no time limit is imposed on interrupt-driven GPIB functions.

Example:
```
C
C Set the driver timeout parameter to 30 seconds.
C
    GPIB(12, 30)
```

SET STATUS

Function Code: 9

Format:
```
RESULT=GPIBU(UNIT,9,S)
          or
RESULT=GPIB(9,S)
```

Remarks: `UNIT` is the interface number. `9` is the function code. `S` is an integer defining the status. The value returned in `RESULT` is 1 if the operation was successful, or is a negative number indicating an error.

The `SET STATUS` function sets the individual status bit (ist) which determines, in part, how the interface responds during a parallel poll conducted by another Controller. If the argument is non-zero, the status bit is set true. If the argument is 0, the status bit is set false.

Example:
```
C
C Set the GPIB interface ist flag to TRUE.
C
    GPIB(9, 1)
```

SPBYTE

Function Code: 16

Format:
```
RESULT=GPIBU(UNIT,16,STAT)
          or
RESULT=GPIB(16,STAT)
```

Remarks: `UNIT` is the interface number. `16` is the function code. Integer `STAT` is the interface serial poll status byte. The value returned in `RESULT` is 1 in all cases.

By using the `SPBYTE` function, you can set or change the status byte that the interface sends when it is serially polled. If the board is not the Controller-In-Charge and bit 0100 octal is set in the status byte, SRQ is asserted as soon as this call is made.

The status byte remains in effect until any one of the following conditions occurs:

- Another call to GPIB `SPBYTE` alters the status byte.
- A call to GPIB `READ` or GPIB `WRITE` is made while the interface is not already addressed. The status byte is then replaced by 0102 or 0104 octal. Refer to the GPIB `READ` function description and the GPIB `WRITE` function description.
- A call to GPIB `FINISH` clears the status byte.
- The interface becomes Controller-In-Charge, in which case the status byte is cleared.

Example:
```
C
C Set the GPIB interface status byte to 0105
C octal, thereby requesting service from the
C current Controller-In-Charge.
C
    GPIB(16, '105'O)
```

STATUS

Function Code: 15

Format:
```
RESULT=GPIBU(UNIT,15,ARRAY,COUNT)
          or
RESULT=GPIB(15,ARRAY,COUNT)
```

Remarks: `UNIT` is the interface number. `15` is the function code. `ARRAY` is a byte array into which the contents of the driver Unit Control Block are written. Integer `COUNT` is the number of bytes to be placed in the buffer. The value returned in `RESULT` is the number of bytes placed in the buffer.

By using the `STATUS` function, you can monitor the state of the driver. The purpose of the call is to place the contents of the interface Unit Control Block into the buffer at address `ARRAY`. If `COUNT` is a value equal to or greater than the actual size of the data area, then the entire data area is placed in the buffer. The actual size and content of the data area varies among different versions of GPIB driver. Detailed knowledge of the driver is generally required in order to interpret the data.

Example:

```
      BYTE V(1000)

      C
      C Read up to 1000 bytes of status information
from
      C the driver structure into V.
      C Return the actual number of bytes read in
      C "RESULT".
      C
          RESULT = GPIB(15, V, 1000)
```

TESTSRQ

Function Code: 13

Format:
```
RESULT=GPIBU(UNIT,13,W)
          or
RESULT=GPIB(13,W)
```

Remarks: `UNIT` is the interface number. `13` is the function code. Integer `W` is the wait flag. A value of 1 in this position causes the driver to suspend itself until the SRQ line is asserted, and then return the value 1. The error `ETIMO` is returned in `RESULT` if SRQ is not asserted within the timeout period (refer to the GPIB `SET PARAMETERS` call). If the `W` argument is 0, the call returns immediately with a 1 to indicate that SRQ is asserted, or `ENONE` to indicate that SRQ is not asserted. Any other negative number returned indicates an error.

The `TESTSRQ` function verifies whether any device is requesting service. If SRQ is asserted, you should perform a `SERIAL POLL` to determine which device is in need of service.

Examples:

```
C
C Suspend execution for 30 seconds while waiting
C for the Service Request (SRQ) line to become
C asserted.
C
    RESULT = GPIB(12, 30)
    RESULT = GPIB(13, 1)
         IF (RESULT.GT.0)
         USER_SERIAL_POLL_ROUTINE
    ELSEIF (RESULT.EQ.ETIMO)
         PRINT *,'No SRQ within timeout period.'
    ELSE
         PRINT *,'Not Controller in Charge!'
    ENDIF
C
C
C Check to see if SRQ has been asserted recently,
C but do not wait around if it hasn't.
C
    IF (GPIB(13, 0).GT.0)
    PRINT *,'SRQ asserted.'
```

TRANSFER

Function Code: 3

Format:
```
RESULT=GPIBU(UNIT,3)
          or
RESULT=GPIB(3)
```

Remarks: UNIT is the interface number. 3 is the function code. The value returned in RESULT is 1 if the operation was successful, or is a negative number indicating an error.

The TRANSFER function allows a transfer of data bytes from a Talking device to one or more Listening devices without having the interface participate. It is assumed that a device has been addressed as Talker and one or more devices have been addressed as Listeners by using a call to the COMMAND function. If the interface is not the Active Controller, an error is returned immediately. Otherwise, the Controller is placed in the stand-by state allowing data to transfer between the addressed devices at the highest possible speed.

The routine returns immediately to the caller and the transfer continues until the next COMMAND, PARALLEL POLL, or CLEAR function call.

Example:

```
C
C Address the device at primary address 3 to
C listen and the device at primary address 4 to
C talk, then go to standby to allow a data
C transfer between the two devices to proceed.
C
    RESULT =GPIB(0, "?_#D", 4)
    RESULT =GPIB(3)
```

WRITE

Function Code: 1

Format:
```
RESULT=GPIBU(UNIT,1,ARRAY,COUNT,MODE)
          or
RESULT=GPIB(1,ARRAY,COUNT,MODE)
```

Remarks: `UNIT` is the interface number. `1` is the function code. `ARRAY` is an array of data bytes to send to all Listening devices. Integer `COUNT` is the size of the array in bytes (0 to 65,535). Integer `MODE` is the write mode (see Table 4-2). If greater than or equal to 0, the value returned in `RESULT` is the number of bytes transferred. A negative returned value indicates an error.

Data messages are sent from the computer to one or more Listening devices only by means of the `WRITE` function. It is assumed that the interface has already been addressed as the Talker and one or more devices have been addressed as Listeners by using a call to the `COMMAND` function, or that another Controller is active and has addressed the board to talk and other devices to listen.

If the interface is the Active Controller when `WRITE` is called, the routine puts the Controller on standby and proceeds to send out data bytes immediately. Otherwise, the routine waits until the interface is addressed as Talker by another Controller and then writes the data at that time. While it is waiting, the routine requests service from the Active Controller (the status byte, octal 102, reflects that it is requesting to talk).

Example:

```
C
C Address the GPIB interface at primary
C address 1 to talk and the two devices at
primary
C addresses 3 and 4 to listen.  Write the data
C string "12345" to both devices and send EOI
C along with the last byte.
C
    RESULT= GPIB(0, "?_A#$", 5)
    RESULT = GPIB(1, "12345", 5, 2)
```

Table 4-2. GPIB WRITE Modes

Mode	Description
0	Terminate on count. Data bytes are output until the requested byte count is reached.
2	Terminate on count with END. Mode 2 is the same as mode 0 but the uniline message, END, is transmitted with the last byte.

Chapter 5
IMICPX

This chapter introduces you to the Interface Bus Interactive Control (IMICPX) program. This chapter also contains instructions for running IMICPX and lists the conventions for using IMICPX.

Introduction to IMICPX

With the Interface Bus Interactive Control (IMICPX) program, you can control the GPIB from the computer keyboard using a convenient input syntax. This program is especially useful for interactively debugging and troubleshooting a complete GPIB system – the bus, the instruments, the interface, and the driver and utility software as well. IMICPX can be used to quickly verify the behavior of an instrument, rather than having to generate several versions of production software for that purpose. If a sequence of calls issued from IMICPX works properly, it is probable (apart from possible timing considerations) that the same sequence of function calls issued from a program will work in the same manner.

Running IMICPX

Start the interactive control program by entering the following command:

```
$ RUN IMICPX                          <Enter>
```

If you changed the Default Bus Table in IBUP.MAR, follow these steps to install IMICPX.

1. Compile the source file by entering the following command:

```
$ FORTRAN /EXTEND_SOURCE IMICPX       <Enter>
```

2. Link the object file with the utility subroutines as follows:

```
$ LINK  IMICPX+IMIBUP+IMGPIB          <Enter>
```

3. Start the interactive control program as follows:

```
$ RUN IMICPX                          <Enter>
```

IMICPX sends its own prompt (:) to the screen and waits for a command specifying the arguments to a GPIB or IBUP function call. Then the results of the function call are printed on the screen and another prompt is sent.

IMICPX remains active until you press <Control-z>, at which time the command prompt re-appears.

When executing functions from within IMICPX, the arguments you use are the same as those described in Chapter 3, *IBUP High-Level Routines*, and Chapter 4, *GPIB Low-Level Routines*, with the following exceptions:

- For the COMMAND and WRITE functions, the arguments ARRAY and COUNT are replaced by a double-quoted string which you supply. The length of the string is automatically counted by IMICPX.
- For the READ and STATUS functions, the argument ARRAY is not required–an input buffer is supplied by IMICPX.

IMICPX Syntax

Figures 5-1 and 5-2 list the format of the input to IMICPX. IMICPX has the following syntax conventions:

- All arguments enclosed in angle brackets (<>) stand for integers.
- All arguments enclosed in double quotation marks (" ") are strings.
- An integer is assumed to be base 10 unless it begins with the digit 0, in which case it is interpreted as base 8; for example, 16 is the same as 020.
- A string is a sequence of characters that is enclosed in double quotation marks; for example, "a string" is a string.
- All arguments are separated by one or more spaces or tabs.
- The square brackets ([]) surround optional characters of a mnemonic; for example, IBUP may be completely written out, or abbreviated as I.
- The function code supplied to IBUP or GPIB may be the mnemonic shown in Figure 5-1, or the corresponding integer value. For example:

  ```
  g cl
  ```

  is equivalent to

  ```
  g 4
  ```

- Upper and lower case characters are treated the same for mnemonics, but are distinct within a string.

Conventions are available for specifying non-printing characters within a string by using the backslash character (\). The format for generating an arbitrary 8-bit byte is \nnn where nnn stands for the 1, 2, or 3 octal digits that define the byte.

Other abbreviations are available for the most common non-printing characters. These special character abbreviations and their meanings are described in Table 5-1.

Table 5-1. IMICPX Non-Printing Character Abbreviations

Abbreviation	Meaning	Equivalent To
\r	carriage return	\15
\n	line feed	\12
\t	tab	\11
\b	backspace	\10

To actually include the backslash character in the string, it is necessary to precede it with a backslash. For example, two backslashes (\\) represent the character \ by itself. A double quotation mark (") can also be included in the string by preceding it with a backslash (\) as in \".

The values returned from the GPIB or IBUP calls are printed in decimal except for valid Parallel Poll and Serial Poll responses, which are printed in octal (the 0400 bit should be ignored). String data from a read call is written directly following the byte count, using the convention for non-printing characters described in the preceding paragraph. The data returned by the status call is printed as 8-bit bytes printed eight to a line, beginning on the line following the returned value.

```
i[bup] w[rite] <dev> "data"
i[bup] rea[d] <dev> <len>
i[bup] cl[ear] <dev>
i[bup] t[rigger] <dev>
i[bup] rem[ote] <dev>
i[bup] l[ocal] <dev>
i[bup] po[ll] <dev>
i[bup] co[nfigure] <dev> <sense> <line>
i[bup] pa[sscontrol] <dev>
i[bup] d[efine] <dev> <tad> <lad> <sad> <rmd> <eod> <wmd>
i[bup] f[inish]
```

Figure 5-1. IMICPX Syntax for High-Level Routines

```
g[pib] co[mmand] "cmds"
g[pib] w[rite] "data" <wmd>
g[pib] r[ead] <len> <rmd> <eod>
g[pib] tr[ansfer]
g[pib] cl[ear]
g[pib] rem[ote]
g[pib] l[ocal]
g[pib] par[allelpoll]
g[pib] pa[sscontrol]
g[pib] sets[tatus] <s>
g[pib] setp[arameters] <t>
g[pib] te[stsrq] <wait>
g[pib] f[inish]
g[pib] st[atus] <count>
g[pib] sp[byte] <byte>
```

Figure 5-2. IMICPX Syntax for Low-Level Routines

Appendix A
Multiline Interface Messages

This appendix lists the multiline interface messages and describes the mnemonics and messages that correspond to the interface functions. These functions include initializing the bus, addressing and unaddressing devices, and setting device modes for local or remote programming. The multiline interface messages are IEEE 488-defined commands that are sent and received with ATN TRUE.

For more information on these messages, refer to the ANSI/IEEE Standard 488.1-1987, *ANSI/IEEE Standard Digital Interface for Programmable Instrumentation.*

Multiline Interface Messages

Hex	Oct	Dec	ASCII	Msg	Hex	Oct	Dec	ASCII	Msg
00	000	0	NUL		20	040	32	SP	MLA0
01	001	1	SOH	GTL	21	041	33	!	MLA1
02	002	2	STX		22	042	34	"	MLA2
03	003	3	ETX		23	043	35	#	MLA3
04	004	4	EOT	SDC	24	044	36	$	MLA4
05	005	5	ENQ	PPC	25	045	37	%	MLA5
06	006	6	ACK		26	046	38	&	MLA6
07	007	7	BEL		27	047	39	'	MLA7
08	010	8	BS	GET	28	050	40	(	MLA8
09	011	9	HT	TCT	29	051	41	)	MLA9
0A	012	10	LF		2A	052	42	*	MLA10
0B	013	11	VT		2B	053	43	+	MLA11
0C	014	12	FF		2C	054	44	,	MLA12
0D	015	13	CR		2D	055	45	-	MLA13
0E	016	14	SO		2E	056	46	.	MLA14
0F	017	15	SI		2F	057	47	/	MLA15
10	020	16	DLE		30	060	48	0	MLA16
11	021	17	DC1	LLO	31	061	49	1	MLA17
12	022	18	DC2		32	062	50	2	MLA18
13	023	19	DC3		33	063	51	3	MLA19
14	024	20	DC4	DCL	34	064	52	4	MLA20
15	025	21	NAK	PPU	35	065	53	5	MLA21
16	026	22	SYN		36	066	54	6	MLA22
17	027	23	ETB		37	067	55	7	MLA23
18	030	24	CAN	SPE	38	070	56	8	MLA24
19	031	25	EM	SPD	39	071	57	9	MLA25
1A	032	26	SUB		3A	072	58	:	MLA26
1B	033	27	ESC		3B	073	59	;	MLA27
1C	034	28	FS		3C	074	60	<	MLA28
1D	035	29	GS		3D	075	61	=	MLA29
1E	036	30	RS		3E	076	62	>	MLA30
1F	037	31	US		3F	077	63	?	UNL

Message Definitions

DCL	Device Clear
GET	Group Execute Trigger
GTL	Go To Local
LLO	Local Lockout
MLA	My Listen Address
MSA	My Secondary Address
MTA	My Talk Address
PPC	Parallel Poll Configure
PPD	Parallel Poll Disable

Multiline Interface Messages

Hex	Oct	Dec	ASCII	Msg	Hex	Oct	Dec	ASCII	Msg
40	100	64	@	MTA0	60	140	96	`	MSA0,PPE
41	101	65	A	MTA1	61	141	97	a	MSA1,PPE
42	102	66	B	MTA2	62	142	98	b	MSA2,PPE
43	103	67	C	MTA3	63	143	99	c	MSA3,PPE
44	104	68	D	MTA4	64	144	100	d	MSA4,PPE
45	105	69	E	MTA5	65	145	101	e	MSA5,PPE
46	106	70	F	MTA6	66	146	102	f	MSA6,PPE
47	107	71	G	MTA7	67	147	103	g	MSA7,PPE
48	110	72	H	MTA8	68	150	104	h	MSA8,PPE
49	111	73	I	MTA9	69	151	105	i	MSA9,PPE
4A	112	74	J	MTA10	6A	152	106	j	MSA10,PPE
4B	113	75	K	MTA11	6B	153	107	k	MSA11,PPE
4C	114	76	L	MTA12	6C	154	108	l	MSA12,PPE
4D	115	77	M	MTA13	6D	155	109	m	MSA13,PPE
4E	116	78	N	MTA14	6E	156	110	n	MSA14,PPE
4F	117	79	O	MTA15	6F	157	111	o	MSA15,PPE
50	120	80	P	MTA16	70	160	112	p	MSA16,PPD
51	121	81	Q	MTA17	71	161	113	q	MSA17,PPD
52	122	82	R	MTA18	72	162	114	r	MSA18,PPD
53	123	83	S	MTA19	73	163	115	s	MSA19,PPD
54	124	84	T	MTA20	74	164	116	t	MSA20,PPD
55	125	85	U	MTA21	75	165	117	u	MSA21,PPD
56	126	86	V	MTA22	76	166	118	v	MSA22,PPD
57	127	87	W	MTA23	77	167	119	w	MSA23,PPD
58	130	88	X	MTA24	78	170	120	x	MSA24,PPD
59	131	89	Y	MTA25	79	171	121	y	MSA25,PPD
5A	132	90	Z	MTA26	7A	172	122	z	MSA26,PPD
5B	133	91	[	MTA27	7B	173	123	{	MSA27,PPD
5C	134	92	\	MTA28	7C	174	124	|	MSA28,PPD
5D	135	93	]	MTA29	7D	175	125	}	MSA29,PPD
5E	136	94	^	MTA30	7E	176	126	~	MSA30,PPD
5F	137	95	_	UNT	7F	177	127	DEL	

PPE	Parallel Poll Enable	SPE	Serial Poll Enable
PPU	Parallel Poll Unconfigure	TCT	Take Control
SDC	Selected Device Clear	UNL	Unlisten
SPD	Serial Poll Disable	UNT	Untalk

Appendix B
Error Code Summary

This appendix contains a list of error codes returned from the driver functions.

Mnemonic	Returned Value	Description
-	> 0	For READ, WRITE, and COMMAND calls, the returned value is the number of bytes transferred in decimal. For PARALLEL POLL and SERIAL POLL calls, the octal poll response ORed with 0400 is returned. For the STATUS call the returned value is the number of bytes stored in the ARRAY.
OK	1	No error.
-	0	Not Used (special internal code).
-	-	No room to allocate string.
ENONE	-1	SRQ not asserted (TEST SRQ).
ECACFLT	-2	ATN remains asserted after IFC sent (bus problem).
ENOTCAC	-3	Not Active Controller for operation requiring CAC (software problem).
ENOTSAC	-4	Not System Controller for operation requiring SAC (software problem).
EIFCLR	-5	IFC caused operation to abort (bus problem).
ETIMO	-6	Operation did not complete within allotted time (bus problem).
ENOFUN	-7	Non-existent driver function code (software problem).
ETCTIMO	-8	Take control not completed within allotted time (bus problem).
ENOIBDEV	-9	No Listeners addressed or no devices connected (bus problem).
EIDMACNT	-10	Internal DMA completed without byte count going to zero (hardware problem).
ENOPP	-11	Parallel Poll operation attempted on three-state GPIB (software problem).
Mnemonic	**Returned Value**	**Description**

EITIMO	-12	Internal register DMA did not complete within allotted time (hardware problem).
EINEXM	-13	Internal register DMA aborted due to non-existent memory (software/hardware problem).
ENEXMEM	-14	GPIB DMA aborted due to nonexistent memory (hardware/software problem).
ECNTRS	-15	Byte address and byte count are inconsistent following GPIB DMA (hardware problem).
ENOUMR	-16	No UNIBUS map register is available for the driver (try again).
EOPEN	-17	IB driver cannot be opened (software problem).
ECADS	-18	User is attempting a Controller Talker or Listener function and the board has previously received the TCT message while addressed as a talker and the old Controller has failed to unassert the ATN line.
—	-19	Not Used.
ENOUFN	-20	Non-existent utility function code (software problem).
ENODEV	-21	Illegal device slot number (software problem).
ENOLAD	-22	No listen address for selected device (software problem).
ENOTAD	-23	No talk address for selected device, or GPIB address in Define call is not legal (software problem).
EHDLR	-100	Communications problem with driver (probably incorrect installation).

Appendix C
FORTRAN Language Call Syntax

This chapter contains a list of FORTRAN language call syntax for high-level and low-level functions.

High-Level Utility Subroutine Summary (IBUP)

Function	Call Syntax
WRITE	RESULT=IBUP(0,D,ARRAY,COUNT)
READ	RESULT=IBUP(1,D,ARRAY,COUNT)
CLEAR	RESULT=IBUP(2,D)
TRIGGER	RESULT=IBUP(3,D)
REMOTE	RESULT=IBUP(4,D)
LOCAL	RESULT=IBUP(5,D)
POLL	RESULT=IBUP(6,D)
CONFIGURE	RESULT=IBUP(7,D,S,L)
PASS CONTROL	RESULT=IBUP(8,D)
DEFINE	RESULT=IBUP(9,D,TAD,LAD,SAD,RMD,EOD,WMD)
FINISH	RESULT=IBUP(10)

Low-Level Utility Subroutine Summary (GPIB)

Function	Call Syntax
COMMAND	RESULT=GPIB(0,ARRAY,COUNT)
WRITE	RESULT=GPIB(1,ARRAY,COUNT,MODE)
READ	RESULT=GPIB(2,ARRAY,COUNT,MODE,EOD)
TRANSFER	RESULT=GPIB(3)
CLEAR	RESULT=GPIB(4)
REMOTE	RESULT=GPIB(5)
LOCAL	RESULT=GPIB(6)
PARALLEL POLL	RESULT=GPIB(7)
PASS CONTROL	RESULT=GPIB(8)
SET STATUS	RESULT=GPIB(9,S)
SET PARAMETERS	RESULT=GPIB(12,T)
TEST SRQ	RESULT=GPIB(13,W)
FINISH	RESULT=GPIB(14)
STATUS	RESULT=GPIB(15,ARRAY,COUNT)
SPBYTE	RESULT=GPIB(16,STAT)

Appendix D
Verifying Hardware and Software Configuration and Installation

This appendix contains instructions for checking the configuration and installation of your interface and the GPIB11 Series VMS software.

Note: If you have the National Instruments GPIB-SCSI interface installed on the DEC VAXstation 3100 or MicroVAX 3100, refer to the getting started manual that came with your hardware for software installation and configuration information.

Initial Verification Using IMICPX

To verify that the software is installed properly (refer to the Getting Started manual that came with your GPIB-SCSI) and that the interface is configured as System Controller (SAC=1), run `IMICPX` (see Chapter 5, *IMICPX*) and perform the actions described in the following steps.

Note: If the interface will not be used as the System Controller after this installation check, reconfigure the necessary software parameters after completing these steps.

Step 1. Send GPIB CLEAR

At the `IMICPX` prompt, enter the following command:

```
g clear                    <Enter>
```

- Expected Results: A value of 1 is returned. If possible, use a bus tester or logic probe to see if the ATN line is asserted.

- Other Results: To interpret error codes, refer to Appendix B, *Error Code Summary*. If the system shuts down or hangs, double-check software parameters configured before compiling. Check the driver file to verify that all compile time parameters have the correct value. Check the *vector* switch configuration on the board.

Step 2. Send GPIB REMOTE

At the IMICPX prompt, enter the following command:

```
g remote                    <Enter>
```

- Expected Results: A value of 1 is returned. If possible, use a bus tester or logic probe to see if the REN line is asserted.

- Other Results: To interpret error codes, refer to Appendix B, *Error Code Summary*.

Step 3. Send GPIB Command

At the IMICPX prompt, enter the following command:

```
g command "?"               <Enter>
```

The ? in double quotation marks represents the Unlisten command. This step requires that a Talker or Listener device known to work properly is connected to the interface to accept the commands. A bus tester or analyzer that is specifically designed to hold up the handshake so that the data lines can be observed is recommended.

- Expected Results: A value of 1 should be returned. If possible, use a bus tester or logic probe to see if the ATN line is asserted and that the correct data lines are asserted.

- Other Results: To interpret error codes, refer to Appendix B, *Error Code Summary*. If the system shuts down or hangs, check the *vector* switch settings on the board and the *vector* value assigned in the software before assembly.

Further IMICPX Testing

After completing the previous steps, use IMICPX to issue any other commands that will be used in your application software. Again, a known working device must be attached to the bus to receive the commands.

Note: The sequence of commands is important for the proper management of the bus, and the sequence of programming bytes within data messages is important for proper control of bus devices. Consult the IEEE 488 specification, Chapter 3, *IBUP High-Level Routines*, and Chapter 4, *GPIB Low-Level Routines*, of this manual, and the user manuals of your bus devices for the correct protocol.

Appendix E
Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve technical problems you might have as well as a form you can use to comment on the product documentation. Filling out a copy of the *Technical Support Form* before contacting National Instruments helps us help you better and faster.

National Instruments provides comprehensive technical assistance around the world. In the U.S. and Canada, applications engineers are available Monday through Friday from 8:00 a.m. to 6:00 p.m. (central time). In other countries, contact the nearest branch office. You may fax questions to us at any time.

Corporate Headquarters
(512) 795-8248
Technical support fax: (800) 328-2203
(512) 794-5678

Branch Offices	Phone Number	Fax Number
Australia	(03) 879 9422	(03) 879 9179
Austria	(0662) 435986	(0662) 437010-19
Belgium	02/757.00.20	02/757.03.11
Denmark	45 76 26 00	45 76 71 11
Finland	(90) 527 2321	(90) 502 2930
France	(1) 48 14 24 00	(1) 48 14 24 14
Germany	089/741 31 30	089/714 60 35
Italy	02/48301892	02/48301915
Japan	(03) 3788-1921	(03) 3788-1923
Netherlands	03480-33466	03480-30673
Norway	32-848400	32-848600
Spain	(91) 640 0085	(91) 640 0533
Sweden	08-730 49 70	08-730 43 70
Switzerland	056/27 00 20	056/27 00 25
U.K.	0635 523545	0635 523154

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name _____

Company _____

Address _____

_____

Fax (___)_____ Phone (___)_____

Computer brand _____ Model _____ Processor _____

Operating system _____

Speed _____MHz RAM _____MB Display adapter _____

Mouse _____yes _____no Other adapters installed _____

Hard disk capacity _____MB Brand _____

Instruments used _____

National Instruments hardware product model _____ Revision _____

Configuration _____

National Instruments software product _____ Version _____

Configuration _____

The problem is _____

_____

_____

_____

_____

List any error messages _____

_____

_____

_____

_____

The following steps will reproduce the problem _____

_____

_____

GPIB11 VMS Software Configuration Form

Record the settings and revisions of your hardware and software on the line to the right of each item. Complete a new copy of this form each time you revise your software or hardware configuration, and use this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

National Instruments Products

- GPIB Hardware _____
- Interrupt Level of Hardware _____
- DMA Channels of Hardware
- Base I/O Address of Hardware _____
- GPIB11 VMS Version _____

Other Products

- Computer Make and Model _____
- Other Boards in System _____
- Base I/O Address of Other Boards _____
- Interrupt Level of Other Boards _____

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: **GPIB11 VMS Software Reference Manual**

Edition Date: **October 1993**

Part Number: **320300-01**

Please comment on the completeness, clarity, and organization of the manual.

If you find errors in the manual, please record the page numbers and describe the errors.

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

_____

Phone (_____) _____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway, MS 53-02
Austin, TX 78730-5039

Fax to: Technical Publications
National Instruments Corporation
MS 53-02
(512) 794-5678

Glossary

Prefix	Meaning	Value
m-	milli-	10^{-3}
μ-	micro-	10^{-6}
n-	nano-	10^{-9}

ANSI	American National Standards Institute
ATN	Attention
CAC	Current Active Controller
CSR	Control-Status Register
DCL	Device Clear
DMA	Direct Memory Access
END	End
EOD	end of data
GET	Group Execute Trigger
GPIB	General Purpose Interface (IEEE 488) Bus
GTL	Go To Local
IBUP	Interface Bus Utility Program
IEEE	Institute of Electrical and Electronic Engineers
IFC	Interface Clear
I/O	input/output
LLO	Local Lockout
MB	megabytes of memory
MLA	My Listen Address
MSA	My Secondary Address
MTA	My Talk Address
PPC	Parallel Poll Configure
PPD	Parallel Poll Disable
PPE	Parallel Poll Enable
PPU	Parallel Poll Unconfigure
REN	Remote Enable
SAC	System Active Controller
SDC	Select Device Clear
SPD	Serial Poll Disable
SPE	Serial Poll Enable
SRQ	Service Request
TCT	Take Control
TRI	Three-State Timing Bit
UNL	Unlisten Command
UNT	Untalk Command
VAX	Virtual Address Extension
VMS	Virtual Memory System