Software Reference Manual

for

National Instruments

GPIB11-Series Interface Cards

January 1987 Edition

Part Number 310001-01

National Instruments
12109 Technology Boulevard
Austin, Texas 78727-6204
(512) 250-9119

User Written Software Drivers

Customers who wish to write their own software handlers for our GPIB interface cards should be aware of the following matters.

The large scale integrated (LSI) circuits used on the GPIB11-2, GPIB11V-1, and the GPIB11V-2 are highly complex and sophisticated. Because of the complexity of the circuits, manufacturers do not provide complete and exhaustive logic tables that describe the behavior of the circuits for every combination of input signals and sequence of input signals. Nor do they provide schematic and timing diagrams so that knowledgeable users can analyze the circuits. Often, even the manufacturer will become aware of an idiosyncrasy only after a user has tried to do something the manufacturer had not tried.

One reason NI provides comprehensive software packages for its interfaces is to overcome the long learning cycle needed to thoroughly understand the behavior of these components. The exact manner in which we implement the high level functions supported by our software is undoubtedly not the only way to implement these functions. However, the software does work, and the hardware we deliver has been tested with this software. NI is responsible for delivering cards that implement specified IEEE Std. 488-1978 interface functions but cannot guarantee that those functions can be implemented by the user using other programming techniques, even if those techniques seem perfectly reasonable.

Customers who write their own software or modify ours - no matter how minor the modification may seem - should be aware that they may be making assumptions about the behavior of the components that may in fact not be true.

Before writing or modifying utility software, it is strongly recommended that users first test the interface with National Instruments unmodified software. This will not only confirm that the hardware/software package performs to specifications, but it will also provide a base from which to proceed in the development of new software. In addition, the user should become thoroughly familiar with this manual, IEEE Std. 488-1978, National Instruments' operating and service manual for the specific interface card, manufacturers' data handbooks for the components used, and programming information of the instruments that will be connected to the GPIB.

TABLE OF CONTENTS

Section Four - Software Installation

Section Five - Interactive Control Program

List of Figures

List of Tables

Section One

INTRODUCTION

The software package delivered with your GPIB11-series interface card consists of two modules as shown in Figure 1.1. The first is the utility routine, which contains information about the instruments connected to the IEEE Std. 488-1975/1978 Standard Digital Interface for Programmable Instrumentation, known as the General Purpose Interface Bus (GPIB). The utility routine uses this information to relieve the applications programmer of some of the addressing protocol inherent in many GPIB OPERATIONS. The utility routine makes use of the second module, the driver routine, to conduct the actual bus I/O. A user program normally calls the utility routine, but, in those circumstances where additional flexibility is required, direct interaction with the driver routine is possible. The driver routine may also be used in a stand-alone environment.

An interactive control program is included in the standard software package. It allows the user to call any of the utility or driver functions from a terminal. This feature enables the user to efficiently check the installation of the hardware and the software. The control program also permits single-step control and operation of the IEEE-488 devices on the bus to aid in the development of application software.

The software in the standard package may be used with the RT and RSX operating systems. Optional software is available for use with the UNIX and the VAX/VMS operating systems. With either of these options, additional reference material is included in the form of a UNIX Addendum or a VAX/VMS Addendum. The utility and driver routines are available as MACRO source files, which may be assembled as FORTRAN-, BASIC-, or MACRO-callable subroutines. In addition, if the GPIB interface card is being used in a system where it is the only GPIB Controller, then code which supports the "pass control" protocol is not assembled, thereby reducing the size of the driver routine as well as the I/O overhead.

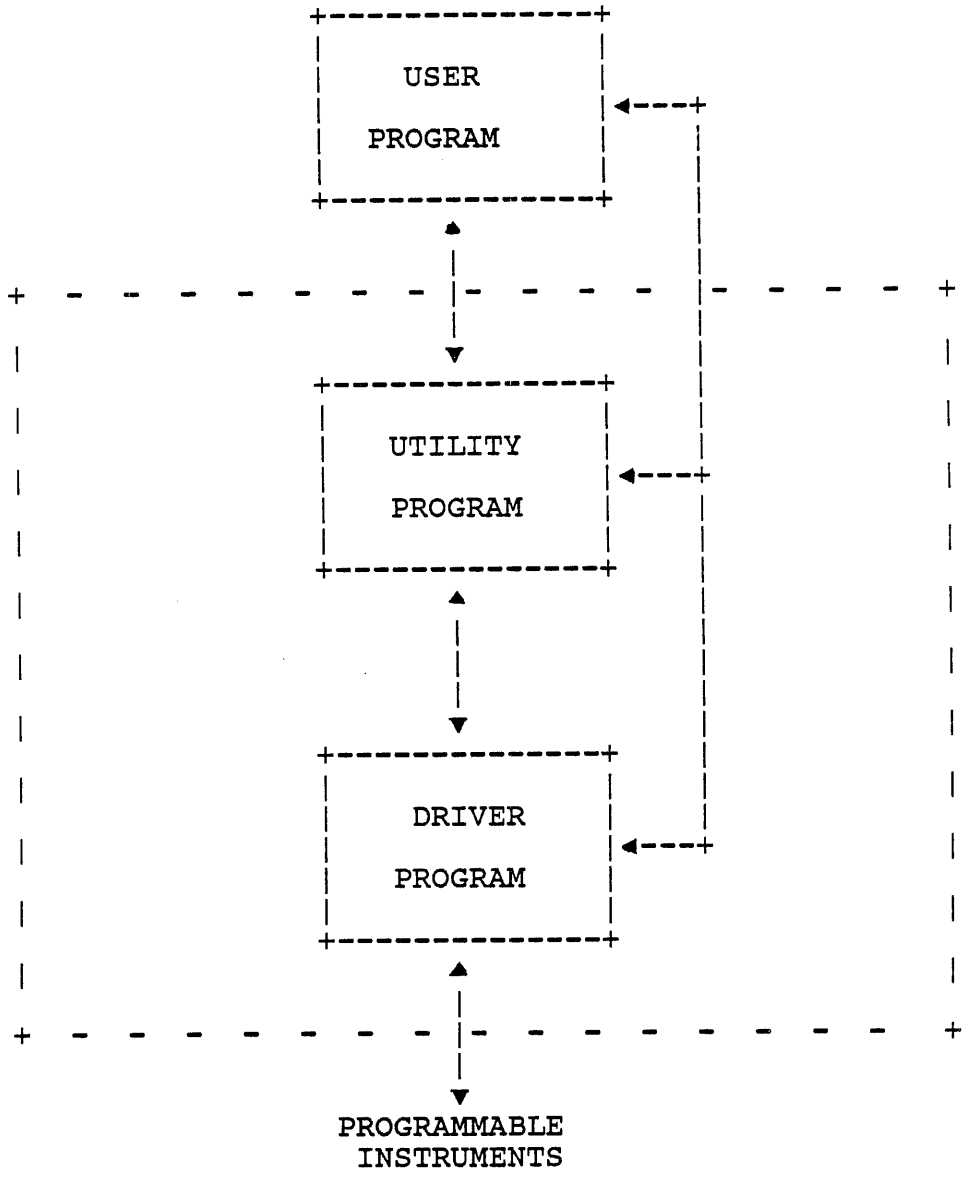

Figure 1.1 - Utility Software

Section Two

UTILITY ROUTINE

2.1 GENERAL

The utility routine allows convenient programming of the GPIB by hiding the complexities of GPIB addressing and protocol and presenting a simple, understandable user interface. The utility routine makes one or more calls on the driver routine in order to perform its functions. Addressing and data transfer information is kept in a device table allowing the user to refer to interfaced instruments by a device table slot number, analogous to a "logical unit number." Some specialized functions that are not provided by the utility routine may be performed by directly calling the driver routine. The user level FORTRAN, MACRO, or BASIC interface to the utility routine is independent of whether the driver routine is installed as a device handler or as a user interrupt routine and is independent of which operating system is used.

The first time the utility routing is called, it automatically issues the driver routine function calls CLEAR and REMOTE, since this is the usual sequence to begin GPIB operation.

NOTE

If the interface is not installed as a System Controller, these calls are ignored.

All utility routine function calls (IBUP) are synchronous, i.e., the return to the calling program is not done until the I/O completes successfully or terminates on an error.

2.2 SYNTAX

The calling syntax for the utility routine functions is described in the following sections.

2.2.1 FORTRAN Call

The calling program must declare IBUP as an integer-valued function. To perform a utility routine function, the calling program executes a FORTRAN function call, as in:

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

FUNCTN is an integer expression that gives the function code, and ARG1 through ARGN are any argument expressions required by the particular function. The value returned by IBUP indicates the success or failure of the operation. Calls with improper function codes return an error.

2.2.2 BASIC CALL

Utility routine functions are invoked using the BASIC CALL statement, as in:

```
CALL "IBUP" (F,A1,...AN).
```

F is an integer expression giving the function code and A1 through AN are any integer or string arguments required by the particular function. No indication is given to the calling program if the call is unsuccessful; however, a message is printed at the terminal in the event of an error. Calls with improper function codes are ignored. All arguments to BASIC are either integers or strings. BASIC syntax for integers requires an appended % as in:

```
CALL "IBUP" (2%,1%).
```

BASIC+2 will return the standard error codes found in Appendix E in the X% argument. The syntax for BASIC+2 as well as BASIC and FORTRAN can be found in Appendix D.

2.2.3 MACRO Call

Calling utility routine functions from an assembly language program is done via:

```
JSR PC, IBUP
```

with R5 pointing to a list of argument references. The standard FORTRAN PC calling convention is used. The utility routine does not and cannot distinguish between a call from a FORTRAN program and a call from an assembly language program. The arguments are the same in both cases.

On return from IBUP, R0 contains the value of the function as defined in the FORTRAN call. The registers R1 through R5 are not preserved across a MACRO call to IBUP.

2.2.4 Device Table

The device table contains an entry for each instrument which is connected to the GPIB. Each entry contains six bytes giving the instrument's talk address, listen address, secondary address, read mode, write mode, and End-Of-Data character. The instrument is then referred to by its location in the device table (slot number) in all calls to the utility routine. Slot 0 is reserved for the GPIB interface card.

The size of the device table is an assembly time parameter. The table may contain unused slots which can then be dynamically modified at run-time. An entry in the device table is a single line of the form:

```
"device TTT, LLL, SSS, R, EOD, W  ;Instrument name"
```

where TTT is the octal talk address (or 0 if the device cannot be a talker); LLL is the octal listen address (or 0 if the device cannot be a listener); SSS is the octal secondary address (or 0 if GPIB 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); and, W is the data transfer mode to use when writing to the device. Refer to Tables 3.1 and 3.2. for descriptions of the write and read data transfer modes, respectively.

Most instruments provide a switch which allows the user to select the lower five bits of the GPIB address to which it will respond. The switch should never be set to all ones since this would conflict with the special GPIB commands Untalk (UNT) and Unlisten (UNL). Any other setting is legal provided each GPIB device has a unique address. To construct a talk address from the five bit switch setting, add octal 100 to the octal switch value. To construct a listen address add octal 40 to the switch value. To construct a secondary address add octal 140 to the switch value.

The following line is an example of a device table entry for a typical instrument whose GPIB address switch has been set to binary 00110 (or octal 006):

```
"device 106, 46, 0, 0, 0, 0 ;typical device"
```

or, equivalently:

```
"device 'F, '&, 0, 0, 0, 0 ;typical device"
```

2.3 FUNCTIONS

The following sections describe the utility routine functions in detail and give example calls in the FORTRAN and BASIC calling syntaxes.

A negative value returned by the function indicates an error. Error conditions and values are described under the referenced driver routine functions in Chapter 3.

In addition to the error codes described with each function, there is a special error code which applies to systems in which the driver routine is installed as a device handler. If an error code of EOPEN(-17.) is returned, it means that the utility routine was unable to access the handler, usually because the handler was not loaded.

IBUP WRITE

IBUP WRITE (function code = 0)

FORTRAN: J=IBUP(0,D,ARRAY,LENGTH)

D is an integer giving the device (slot) number. ARRAY is an array of bytes to be written to the device. LENGTH is an integer giving the size of the array in bytes. The returned value is the number of bytes transferred if successful or the code returned by the GPIB COMMAND function or WRITE function if an error occurs.

BASIC: CALL "IBUP"(0%,D%,W$)

D is an integer giving the device (slot) number. W$ is a string of bytes to be written to the device.

The WRITE function addresses the selected device as Listener and transfers the data bytes from the computer to that device using the write mode specified in the device table (see Table 3.1).

The WRITE function calls the GPIB COMMAND function to perform the addressing and the GPIB WRITE function to transmit the data, and the GPIB COMMAND to perform the unaddressing. For more details, refer to the GPIB driver routine description.

IBUP READ

IBUP READ (function code = 1)

FORTRAN: J=IBUP(1,D,ARRAY,COUNT)

D is an integer giving the device (slot) number. ARRAY is a byte array in which bytes received from the device are placed. COUNT is an integer giving the number of bytes to read. The returned value is the actual number of bytes read for a successful read and the code returned by the GPIB COMMAND function or READ function if an error occurs.

BASIC: CALL "IBUP"(1%,D%,R$,C%)

D is an integer giving the device (slot) number. R$ is a string variable in which bytes received from the device are placed. C is an integer giving the number of bytes to read.

The READ function addresses the selected device as Talker and transfers the data bytes from that device to the computer using the read mode specified in the device table (see Table 3.2).

The READ function calls the GPIB COMMAND function to perform the addressing and the GPIB READ function to input the data. For more details, refer to the GPIB driver routine description.

IBUP CLEAR

IBUP CLEAR (function code = 2)

FORTRAN: J=IBUP(2,D)

D is an integer giving the device (slot) number. The rettuned value is 1 unless an error occurred, in which case it is the code returned by the GPIB COMMAND function or CLEAR function.

BASIC: CALL "IBUP" (2%,D%)

D is an integer giving the device (slot) number.

The CLEAR function is used to selectively clear one device on the GPIB, to simultaneously clear all devices on the GPIB, or to initialize the GPIB itself by sending the Interface Clear (IFC) message. If the device (slot) number is 0 all devices are cleared (via the DCL interface message); if it 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 the addressing and sending of 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 driver routine description.

IBUP TRIGGER

IBUP TRIGGER (function code = 3)

FORTRAN: J=IBUP(3,D)

D is an integer giving the device (slot) number. The returned value is 1 unless an error occurred, in which case it is the code returned by the GPIB COMMAND function.

BASIC: CALL "IBUP"(3%,D%)

D is an integer giving the device (slot) number.

The TRIGGER function is used to selectively trigger one device or to simultaneously trigger all devices. If the device (slot) number is 0 all devices are triggered; otherwise, only the selected device is triggered.

The TRIGGER function calls the GPIB COMMAND function to perform the addressing and sending of the Group Execute Trigger (GET) message. For more details refer to the GPIB driver routine description.

IBUP REMOTE

IBUP REMOTE (function code = 4)

FORTRAN: J=IBUP(4,D)

D is an integer giving the device (slot) number. The returned value is 1 unless an error occurred, in which case it is the code returned by the GPIB COMMAND or REMOTE functions.

BASIC: CALL "IBUP"(4%,D%)

D is an integer giving the device (slot) number.

The REMOTE function is used to put a selected device in remote mode, to put all devices in remote mode, or to put all devices in remote mode with local lockout. If the device (slot) number is 0, all devices are put in remote mode; if it is negative, all devices are put in remote mode with Local Lockout; otherwise, only the selected device is put in remote mode. Unless the local lockout mode is used, an interfaced device may override the remote program control by means of a front panel local mode switch.

The REMOTE function calls the GPIB COMMAND function to perform the addressing and sending of 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 driver routine description.

IBUP LOCAL

IBUP LOCAL (function code = 5)

FORTRAN: J=IBUP(5,D)

D is an integer giving the device (slot) number. The returned value is 1 unless an error occurred, in which case it is the code returned by the GPIB COMMAND or LOCAL functions.

BASIC: CALL "IBUP"(5%,D%)

D is an integer giving the device (slot) number.

The LOCAL function is used to return a selected device to local mode, to return all devices to local mode, or to return all devices to local mode and cancel the Local Lockout message. If the device (slot) number is 0, all devices are returned to local mode; if it is negative, all devices are returned to local mode and the Local Lockout message is cancelled; otherwise, only the selected device is returned to local mode.

The LOCAL function calls the GPIB COMMAND function to perform addressing and sending of 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 driver routine description.

IBUP POLL (function code = 6)

FORTRAN: J=IBUP(6,D)

D is an integer giving the device (slot) number. The returned value is the status byte (or parallel poll byte) unless an error occurred, in which case it is the code returned by the GPIB COMMAND, READ, or PARALLEL POLL functions.

BASIC: CALL "IBUP"(6%,D%,P%)

D is an integer giving the device (slot) number. P is the integer variable in which the status byte (or parallel poll byte) is returned.

The POLL function is used to serially poll a selected device, to poll all devices in parallel, or to test if any device is requesting service. If the device (slot) number is 0, all devices are polled in parallel. If it is negative, the status of the SRQ line is returned (which is -1 if no device is requesting service and 1 if at least one device is requesting service and the GPIB interface card is the active controller; i.e., 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 and the GPIB PARALLEL POLL function to conduct a parallel poll. Note that the parallel poll byte is logically "OR"ed with octal 400 before being returned. For more details, refer to the GPIB driver routine description.

IBUP CONFIGURE (function code = 7)

FORTRAN: J=IBUP(7,D,S,L)

D is an integer giving the integer device (slot) number. S is an integer giving the sense of the response. L is an integer giving the data line for the response. The returned value is 1 unless an error occurred, in which case it is the code returned by the GPIB COMMAND function.

BASIC: CALL "IBUP"(7%,D%,S%,L%)

D is an integer giving the device (slot) number. S is an integer giving the sense of the response. L is an integer giving the data line for the response.

The CONFIGURE function is used to configure a selected device for parallel polling or to unconfigure all devices. If the device (slot) number is 0, all devices are unconfigured; otherwise, the selected device is configured to respond on line L when its individual status bit (ist) is true (S>0) or to respond when its ist bit is false (S=0). If S is negative, the selected device is unconfigured. 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 sending of the Parallel Poll Enable (PPE) and Parallel Poll Disable (PPD) messages and sending of the Parallel Poll Configure (PPC) and Parallel Poll Unconfigure (PPU) messages. For more details, refer to the GPIB driver routine description.

IBUP PASS CONTROL

IBUP PASS CONTROL (function code = 8)

FORTRAN: J=IBUP(8,D)

D is an integer giving the device (slot) number. The returned value is 1 unless an error occurred, in which case it is the code returned by the GPIB COMMAND function.

BASIC: CALL "IBUP" (8%,D%)

D is an integer giving the device (slot) number.

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

The PASS CONTROL function calls the GPIB COMMAND function to perform all the addressing and sending of the Take Control (TCT) message. The GPIB PASS CONTROL function is called to let the selected device become the Controller-In-Charge. For more details, refer to the GPIB driver routine description.

IBUP DEFINE (function code = 9)

FORTRAN: J=IBUP(9,D,TAD,LAD,SAD,RMD,EOD,WMD)

D is an integer giving the device (slot) number. TAD is an integer giving the GPIB Talk address (0100 to 0136 octal, or 0 if the device is not a Talker). LAD is an integer giving the GPIB Listen address (040 to 076 octal, or 0 if the device is not a Listener). SAD is an integer giving the GPIB Secondary address (0140 to 0176 octal, or 0 if the device does not have a Secondary address). RMD and WMD are integers giving the read and write modes, respectively. (See Tables 3.1 and 3.2.) EOD is an integer giving the End-Of-Data character for certain read modes. The returned value is a 1 to indicate success or a negative number to indicate an error.

BASIC: CALL "IBUP"(9%,D%,T%,L%,S%,R%,E%,W%)

D is an integer giving the device (slot) number. T is an integer giving the GPIB Talk address (0100 to 0136 octal, or 0 if the device is not a Talker). L is an integer giving the GPIB Listen address (040 to 076 octal, or 0 if the device is not a Listener). S is an integer giving the GPIB Secondary address (0140 to 0176 octal, or 0 if the device does not have a Secondary address). R, E, and W are integers giving the read mode, End-Of-Data character, and write mode respectively. (See Tables 3.1 and 3.2.)

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

IBUP FINISH

IBUP FINISH (function code = 10)

FORTRAN: J=IBUP(10)

BASIC: CALL "IBUP"(10%)

The FINISH function is used to terminate usage of the GPIB by calling the GPIB FINISH function routine to disable interrupts and re-initialize the interface and to close the I/O channel if necessary. The returned value is a 1 to indicate success or a negative number to indicate an error.

Section Three

DRIVER ROUTINE

3.1 GENERAL

The driver routine provides the complete flexibility to accomplish all GPIB operations but requires the user to be familiar with the IEEE Std. 488-1978 specification. In most applications, the utility routine provides all the functions which are needed; however, in certain circumstances, additional flexibility is required, necessitating a direct call on the driver routine by the user program. Such calls may be freely intermixed with calls upon the utility routine.

The driver routine may be

* installed as a handler under the RSX operating system,
* installed as a handler under the RT operating system, or
* linked under RT to the user applications program to run under RT or to run stand-alone under no operating system.

When the driver is installed as handler, the function calls (GPIB) are synchronous, i.e., the return to the calling program is not done until the I/O completes successfully or terminates on an error. When the driver is linked to the applications program, either synchronous or asynchronous I/O is selected at configuration time. With asynchronous operation, the calling program initiates I/O, continues processing, and returns at a later time to wait for completion or termination of the I/O. See Section 4.3. For most applications, synchronous I/O is recommended.

3.2 SYNTAX

The calling syntax for the driver program functions is described in the following sections.

3.2.1 FORTRAN Call

The calling program must declare GPIB as an integer valued function. To perform a driver program function, the calling program executes a FORTRAN function call as in:

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

FUNCTN is an integer expression which gives the function code and ARG1 through ARGN are any integer or array argument expressions required by the particular function. The integer value returned by GPIB indicates the success or failure of the operation. Calls with improper function codes return an error.

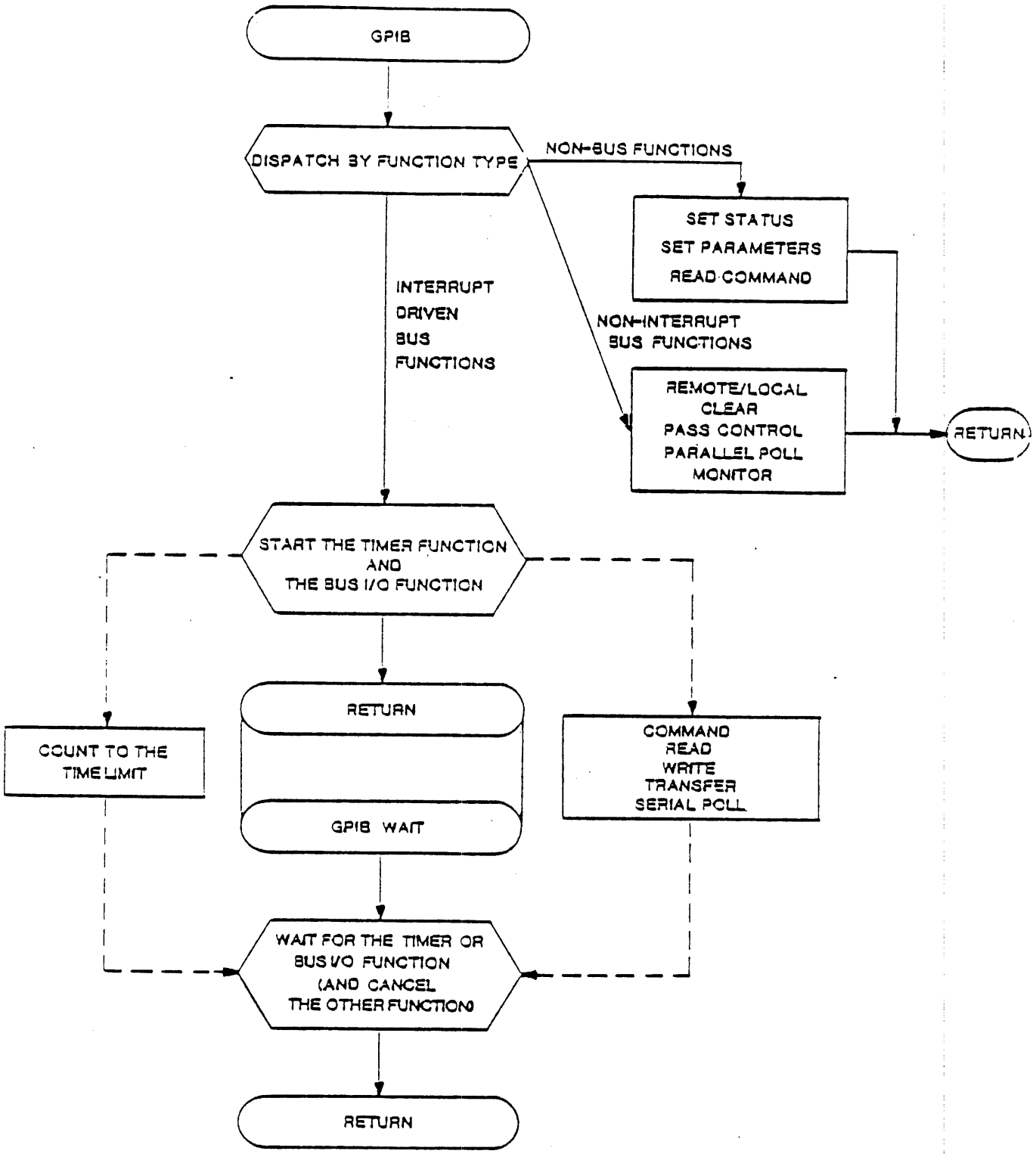

Figure 3.1 - Driver Program Block Diagram

3.2.2 BASIC Call

Driver program functions are invoked using the BASIC CALL statement as in:

```
CALL "GPIB"(F,A1,...,AN).
```

F is an integer expression giving the function code, and A1 through AN are any integer or string arguments required by the particular function. No indication is given to the calling program if the call is unsuccessful. If unsuccessful, the user is notified by a message at the terminal. Calls with improper function codes are ignored. Note that the BASIC syntax for integers requires an appended % as in:

```
CALL "GPIB"(0%,C$).
```

BASIC+2 will return the standard error codes found in Appendix E in the X% argument. The syntax for BASIC+2 as well as BASIC and FORTRAN can be found in Appendix D.

3.2.3 Macro Call

Calling driver routine functions from an assembly language program is done via:

```
JSR PC,GPIB
```

with R5 pointing to a list of argument references; i.e., the standard FORTRAN PC calling convention is used. The driver routine does not and cannot distinguish between a call from a FORTRAN program and a call from an assembly language program. The arguments are the same in both cases.

On return from GPIB, R0 contains the value of the function as in the FORTRAN call. The registers R1 through R5 are preserved across a call to GPIB.

3.3 FUNCTIONS

The following sections describe the driver routine functions in detail, especially with regard to the interaction with the interface card's Controller function; i.e., whether or not the interface card is the System Controller and/or the Active Controller.

A negative value returned by the function indicates an error. Error codes and descriptions are in Appendix E.

GPIB COMMAND

GPIB COMMAND (function code = 0)

FORTRAN: J=GPIB(0,ARRAY,LENGTH)

ARRAY is an array of command bytes to be sent out on the GPIB. LENGTH is an integer giving the size of the array in bytes. The returned value is the number of bytes transferred if successful or a negative number indicating an error.

BASIC: CALL "GPIB"(0%,C$)

C$ is a string of command bytes to be sent out on the GPIB. If an error occurs, a message is printed and the routine returns to BASIC immediate mode.

The COMMAND function is the only means by which multiline command messages may be sent on the GPIB. The utility 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 other multiline command messages: DCL, GET, GTL, LLO, PPC, PPD, PPE, PPU, SDC, SPD, SPE, and TCT.

If the interface card is the Active Controller, the command messages are output immediately. Otherwise, the program will wait until it becomes Active Controller (by receiving control from another Controller) and output the messages at that time. While waiting, the interface requests service from the current Active Controller (the status byte, octal 101, reflects that it is requesting control be passed back).

If the interface card is configured as the System Controller, the CLEAR function must be called prior to the first call to COMMAND.

GPIB WRITE

GPIB WRITE (function code = 1)

FORTRAN: J=GPIB(1,ARRAY,LENGTH,MODE)

ARRAY is an array of data bytes to be sent to all Listening devices. LENGTH is an integer giving the size of the array in bytes. MODE is an integer specifying the output format. The returned value is the number of bytes written to signal success or a negative number indicating an error.

BASIC: CALL "GPIB"(1%,W$,M%)

W$ is a string of data bytes to be sent to all Listening devices. M is an integer specifying the output format. If an error occurs, a message is printed and the routine returns to BASIC immediate mode.

The WRITE function is the only means by which data messages may be sent from the computer to one or more Listening devices. It is assumed that the GPIB interface card 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 done the GPIB addressing.

If the interface card is the Active Controller when WRITE is called, the driver routine will put the Controller on standby and proceed to send out data bytes immediately. Otherwise, if the card is not the Active Controller, the routine will wait until the GPIB interface card is addressed as Talker by another Controller and output the data bytes at that time. While it is waiting, the driver routine requests service from the Active Controller (the status byte, octal 102, reflects that it is requesting to talk).

The output modes are described in Table 3.1.

Table 3.1 - Write Modes

Mode	Description
0	Terminate on count. Data bytes are output unmodified until the requested byte count is reached.
2	Terminate on count with END. Same as mode 0 but the uniline message, END,, is transmitted with the last byte. END is the message received when the EOI (End of Identify) line is asserted during data transfers, i.e., when ATN is false.

GPIB READ

GPIB READ (function code = 2)

FORTRAN: J=GPIB(2,ARRAY,LENGTH,MODE,EOD)

ARRAY is an array in which bytes read from the Talker are placed. LENGTH is an integer giving the number of bytes to read. MODE is an integer specifying the input format. EOD is an integer containing the End-Of-Data character. The returned value is equal to the actual number of data bytes read if the operation was successful or a negative number indicating an error.

BASIC: CALL "GPIB"(2%,R$,L%,M%,E%)

R$ is a string variable in which bytes read from the Talker are placed. L is an integer giving the number of bytes to read. M is an integer specifying the input format. E is an integer containing the End-Of-Data character. If an error occurs, a message is printed and the routine returns to BASIC immediate mode.

The READ function is the only means by which data bytes may be sent from a Talking device to the computer. 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 done the GPIB addressing.

If the interface card is the Active Controller when READ is called, the driver routine will enable the programmable listen function, put the Controller on standby, and begin reading data bytes. Otherwise, the driver routine will wait until the GPIB interface card is addressed as Listener and input the data bytes at that time. While it is waiting, the driver routine requests service from the current Active Controller (the status byte, octal 104, reflects that it is requesting to Listen).

The input modes are described in Table 3.2.

Table 3.2 - 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. END is the message received when the EOI (End or Identify) line is asserted during data transfers, i.e., when ATN is false.
2	Terminate on count or END or EOD. Same as mode 0 except reading is also terminated when a byte is read which is equal to End-Of-Data (EOD) character.
4	Terminate on count only. Data bytes are read until the requested byte count is reached. END or EOD will not terminate the read.
* 6	Terminate on count or EOD. Data bytes are read until the requested byte count is reach, or until a byte is read which is equal to the value of EOD. END will not terminate the read.

* Mode 6 is not available on DMA boards, i.e., END cannot be selectively ignored when EOD is recognized.

GPIB TRANSFER

GPIB TRANSFER (function code = 3)

FORTRAN: J=GPIB(3)

The returned value is 1 if the operation was successful or a negative number indicating an error.

BASIC: CALL "GPIB"(3%)

If an error occurs, a message is printed, and the routine returns to BASIC immediate mode.

The TRANSFER function is used to allow a transfer of data bytes from a Talking device to one or more Listening devices without having the computer 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 card is not the Active Controller, an error is returned immediately. Otherwise, the Controller is placed in the stand-by state allowing data transfer between the addressed devices at the highest speed possible.

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

GPIB CLEAR

GPIB CLEAR (function code = 4)

FORTRAN: J=GPIB(4)

The returned value is 1 if the interface card is the System Controller or a negative number indicating an error.

BASIC: CALL "GPIB"(4%)

If an error occurs or the interface card is not the System Controller, a message is printed and the routine returns to BASIC immediate mode.

The CLEAR function is used to initialize the GPIB by sending the IFC message. It returns an error unless the interface card is configured as the System Controller. This function is used to activate the Controller-In-Charge function of the System Controller and to recover from a GPIB error.

GPIB REMOTE

GPIB REMOTE (function code = 5)

FORTRAN: J=GPIB(5)

The returned value is 1 if the interface card is the System Controller or a negative number indicating an error.

BASIC: CALL "GPIB"(5%)

If an error occurs or the interface card is not the System Controller, a message is printed and the routine returns to BASIC immediate mode.

The REMOTE function is used when placing devices in remote mode. It returns an error if the interface card is not configured as the System Controller. REMOTE sends the REN message; however, devices on the GPIB 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.

GPIB LOCAL

GPIB LOCAL (function code = 6)

FORTRAN: J=GPIB(6)

The returned value is 1 if the interface card is the System Controller or a negative number indicating an error.

BASIC: CALL "GPIB"(6%)

If an error occurs or the interface card is not the System Controller, a message is printed and the routine returns to BASIC immediate mode.

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

GPIB PARALLEL POLL

GPIB PARALLEL POLL (function code = 7)

FORTRAN: J=GPIB(7)

The returned value is the Parallel Poll result or a negative number indicating an error.

BASIC: CALL "GPIB"(7%,P%)

The Parallel Poll result is returned in P. If an error occurs, the routine returns to BASIC immediate mode.

The PARALLEL POLL function is used to poll the interfaced devices which have been configured for parallel polling. A device may be dynamically configured by using the COMMAND function or statically configured by a switch on the device.

If the interface card is the Active Controller, the program will take the poll immediately and return the results. Otherwise, the program will return an error.

The Parallel Poll result is the 8 bit response which appeared on the GPIB DIO lines, logically "OR"ed with octal 400.

GPIB PASS CONTROL

GPIB PASS CONTROL (function code = 8)

FORTRAN: J=GPIB(8)

The returned value is 1 if the interface card is the Active Controller on entry or a negative number indicating an error.

BASIC: CALL "GPIB"(8%)

If an error occurs or the interface card is not the Active Controller, a message is printed and the routine returns to BASIC immediate mode.

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

When another COMMAND, WRITE, or READ call is made to the GPIB driver routine and control has not been returned, service is requested from the current Active Controller (the status byte which is presented when the interface card is serially polled is octal 101, 102, or 104, for COMMAND, WRITE, or READ, respectively). If the interface card is the System Controller, it may take control from any other Controller at any time by calling the CLEAR function.

GPIB SET STATUS

GPIB SET STATUS (function code = 9)

FORTRAN: J=GPIB(9,S)

S is an integer specifying the status. The returned value is always 1 or a negative number indicating an error.

BASIC: CALL "GPIB"(9%,S%)

S is an integer specifying the status. If an error occurs, the routine returns to BASIC immediate mode.

The SET STATUS function is used to set the individual status bit (IST) which is used in the response to a Parallel Poll. If the argument is non-zero, the status bit is set true. If the argument is zero, the status bit is set false.

GPIB MONITOR

GPIB MONITOR (function code = 10)

FORTRAN: J=GPIB(10,M)

M is an integer specifying MONITOR mode ON or Off. The returned value is always 1 or a negative number indicating an error.

BASIC: CALL "GPIB"(10%,M%)

M is an integer specifying MONITOR mode ON or OFF. If an error occurs, the routine returns to BASIC immediate mode.

The MONITOR function is used to allow the interface card to read and monitor command bytes which are sent by another Controller while the interface card is not the Active Controller. If the argument to MONITOR is zero, then monitoring is disabled. If the argument is non-zero, monitoring is enabled, and applicable command bytes are automatically read from the GPIB and placed in a 32 byte FIFO buffer. Command bytes are read from the buffer using the READ COMMAND function (see GPIB READ COMMAND).

Monitoring command bytes is not a standard GPIB operation. The command bytes that can be monitored depend on which GPIB interface card, the type and group of message, and the address state of the interface card.

GPIB READ COMMAND

GPIB READ COMMAND (function code = 11)

FORTRAN: J=GPIB(11)

The returned value is the next command byte that was monitored, -1 if no command byte is available, or another negative number if an error occured.

BASIC: CALL "GPIB"(11%,C%)

C is an integer variable in which the next command byte that was monitored is placed. If no byte is available, -1 is returned in C. If an error occurs, the routine returns to BASIC immediate mode.

The READ COMMAND function is used in conjunction with the MONITOR function to read command bytes sent out on the GPIB by another Controller.

If a command byte is not available if the FIFO buffer, -1 is returned so that the user program may loop, waiting for a command, or continue processing and call later to check for a command byte. Command bytes are placed in 32 byte internal buffer. If the buffer fills, subsequent GPIB command bytes are lost until buffer space becomes available (no indication of lost bytes is available).

GPIB SET PARAMETERS (function code = 12)

FORTRAN: J=GPIB(12,T)

T is an integer giving the time limit in seconds. The returned value is always 1 or a negative number if an error occurred.

BASIC: CALL "GPIB"(12%,T%)

T is an integer giving the time limit in seconds. If a error occurs, the routine returns to BASIC immediate mode.

The SET PARAMETERS function is used to establish the driver routine 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=0, no time limit is imposed on interrupt-driven GPIB functions.

GPIB TEST SRQ

GPIB TEST SRQ (function code = 13)

FORTRAN: J=GPIB(13,W)

If W is non-zero, the function waits until SRQ is asserted. Otherwise, the function returns immediately. The returned value is -1 if SRQ is not being asserted and 1 if one or more devices are asserting SRQ. Any other negative number indicates an error.

BASIC: CALL "GPIB"(13%,S%,W%)

S is an integer variable which is set to 1 if SRQ is asserted and set to -1 if SRQ is not asserted. If an error occurs, this routine returns to BASIC immediate mode.

The TEST SRQ function is used to check if any devices are requesting service. If SRQ is found to be asserted, a Serial Poll is usually done to determine which device is in need of service.

GPIB FINISH

GPIB FINISH (function code = 14)

FORTRAN: J=GPIB(14)

BASIC: CALL "GPIB"(14%)

The FINISH function disables all interface card interrupts and initializes the interface card. The return value is 1.

GPIB[U] STATUS

GPIB[U] STATUS (function code = 15)

```
FORTRAN:  J=GPIBU(UNIT,15,ARRAY,COUNT)
          J=GPIB(15,ARRAY,COUNT)

BASIC:    CALL "GPIBU"(UNIT%,15%,ARRAY%,COUNT%)
          CALL "GPIB"(15%,ARRAY%,COUNT%)
```

This call enables the user to monitor the state of the driver. The effect of the call is to place the contents of the interface nit Control Block into the buffer at address ARRAY. UNIT is the interface board unit number. Integer COUNT is the number of bytes to be placed in the buffer. If a COUNT equal to or greater than the actual size of the data area is used, then the entire data area is placed in the buffer. The actual size and content of the data area varies with each type of interface board. Detailed knowledge of the driver is generally required in order to interpret the data.

GPIB[U] SPBYTE

GPIB[U] SPBYTE (function code = 16)

```
FORTRAN:  J=GPIBU(UNIT,16,STAT)
          J=GPIB(16,STAT)

BASIC:    CALL "GPIBU"(UNIT%,16%,STAT%)
          CALL "GPIB"(16%,STAT%)
```

This function enables the user to set or change the status byte that the interface will send when it is serially polled. UNIT is the interface board unit number. Integer STAT is the interface board serial poll status byte. 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 occurs:

* Another call to GPIB[U] SPBYTE alters the status byte.

* A call to GPIB[U] READ or GPIB[U] WRITE is made while the interface is not already addressed. The status byte is then replaced by 0102 or 0104 (refer to GPIB[U] READ, GPIB[U] WRITE).

* A call to GPIB[U] FINISH clears the status byte.

* The interface becomes Controller-In-Charge, in which case the status byte is cleared.

Section Four

SOFTWARE INSTALLATION

The files and procedures to use for installing the GPIB software depend on the interface card that is used, the target operating system, and the language support desired. The many options available make the description of the installation necessarily detailed; however, you will quickly see that for a given software and hardware configuration the actual effort involved is minimal. The following steps should be followed to install the proper software for a single interface card on your system. Contact the factory for special instructions if you are installing more than one GPIB interface card in your system. For UNIX or VAX software installation, see the appropriate addendum. For an overview of the entire installation process, refer to Appendix F.

4.1 STEP 1: REQUIRED DISTRIBUTION FILES

Table 4.1 lists the files which are needed for each configuration. Locate the operating system you will be using along the top row and the interface card you will be using down the left column. At the intersection is the list of the files required for installation. These files should be copied from the distribution medium to a suitable working area. The files of the original distribution should be left undisturbed so that they can be used for later reference, or for restarting the installation procedure. In addition to the files listed in Table 4.1 the file IBSCLI.MAC is used for all configurations that require BASIC support. A brief description of each file is given in Table 4.2.

Note that the stand-alone modules may actually be used under RT as a user interrupt routine in place of the RT handler. (The OS assembly parameter indicates whether the software is to run under RT or stand-alone. And the DVR assembly parameter indicates whether it is to run with a handler or not, as explained in Section 4.3.) This is the recommended installation under RT because RT is not a multi-user operating system. It is not possible to use the stand-alone modules under RSX because they do not support the RSX connect-to-interrupt feature.

4.2 STEP 2: FILE NAMING CONVENTION

If the software is being installed as an RSX or RT handler, the files must be renamed. Under both operating systems device handlers are restricted to two-letter mnemonics. The file names should be changed as shown in Table 4.3.

The stand-alone version of the driver routine does not need to be renamed. For simplicity, in the following descriptions the name IBDP.MAC will be used to refer to the appropriate stand-alone driver routine IBDP.MAC, IBVDP.MAC, IB2DP.MAC, or IBV2DP.MAC.

Table 4.1 - Files Required for Installation

Interface	Operating System (and Installation)			
	RSX-11 (handler)	RT-11, TSX (handler)	RT-11 (linked routine)	No OS stand-alone (linked routine)
GPIB11-1	IBUP.MAC IBUDP.MAC IBDRV.MAC IBTAB.MAC	IBUP.MAC IBUDP.MAC IB.MAC	IBUP.MAC IBDP.MAC	IBUP.MAC IBDP.MAC
GPIB11V-1	IBUP.MAC IBUDP.MAC IBVDRV.MAC IBVTAB.MAC	IBUP.MAC IBUDP.MAC IBV.MAC	IBUP.MAC IBVDP.MAC	IBUP.MAC IBVDP.MAC
GPIB11-2	IBUP.MAC IBUDP.MAC IB2DRV.MAC IB2TAB.MAC	IBUP.MAC IBUDP.MAC IB2.MAC	IBUP.MAC IB2DP.MAC	IBUP.MAC IB2DP.MAC
GPIB11V-2	IBUP.MAC IBUDP.MAC IV2DRV.MAC IV2TAB.MAC	IBUP.MAC IBUDP.MAC IBV2.MAC	IBUP.MAC IBV2DP.MAC	IBUP.MAC IBV2DP.MAC

Note: IBSCLI.MAC is required for BASIC support.

Table 4.2 - Source File Description

IBUDP.MAC	Handler interface routine that provides the GPIB() functions under RT and RSX.
IBUP.MAC	Utility routine providing IBUP() calls for all installations (conditional assembly code selects installation option)
IBDP.MAC	GPIB11-1 stand-alone driver routine providing GPIB() calls
IBVDP.MAC	GPIB11V-1 stand-alone driver routine providing GPIB() calls.
IB2DP.MAC	GPIB11-2 stand-alone driver routine providing GPIB() calls.
IBV2DP.MAC	GPIB11V-2 stand-alone driver routine providing GPIB() calls.
IB.MAC	GPIB11-1 RT Device Handler
IBV.MAC	GPIB11V-1 RT Device Handler
IB2.MAC	GPIB11-2 RT Device Handler
IBV2.MAC	GPIB11V-2 RT Device Handler
IBDRV.MAC IBTAB.MAC	GPIB11-1 RSX Device Driver
IBVDRV.MAC IBVTAB.MAC	GPIB11V-1 RSX Device Driver
IB2DRV.MAC IB2TAB.MAC	GPIB11-2 RSX Device Driver
IV2DRV.MAC IV2TAB.MAC	GPIB11V-2 RSX Device Driver
IBSCLI.MAC	BASIC Function Table (for use in re-linking BASIC when the routines are conditionally assembled for use with BASIC)

Table 4.3 - Source File Renaming

Distribution Name	RT New Name	RSX New Name
IB.MAC	IB.MAC	N/A
IBV.MAC	IB.MAC	N/A
IB2.MAC	IB.MAC	N/A
IBV2.MAC	IB.MAC	N/A
IBDRV.MAC	N/A	IBDRV.MAC
IBTAB.MAC	N/A	IBTAB.MAC
IBVDRV.MAC	N/A	IBDRV.MAC
IBVTAB.MAC	N/A	IBTAB.MAC
IB2DRV.MAC	N/A	IBDRV.MAC
IB2TAB.MAC	N/A	IBTAB.MAC
IV2DRV.MAC	N/A	IBDRV.MAC
IV2TAB.MAC	N/A	IBTAB.MAC

4.3 STEP 3: ASSEMBLY PARAMETER EDITING

If the default UNIBUS address and vector address have been changed or if the default assembly parameters are unsuitable, then the driver file (e.g., IBDRV.MAC and IBTAB.MAC, or IB.MAC, or IBDP.MAC) must be edited before proceeding. Table 4.4 lists the driver assembly parameters, their meaning, and their default values for each interface. The driver assembly parameters are located at the beginning of the file.

The linked drivers have two assembly parameters, ASYNCH and DIAG, that should normally be left with their default values (0). Setting ASYNCH to a non-zero value causes GPIB to return immediately without waiting for the operation to complete. In this case, if a non-zero value is returned by GPIB, it means the operation did complete (negative for error, positive otherwise). A zero result means the operation is in progress, and the function IBWAIT must be called prior to issuing another GPIB call. The value returned by IBWAIT is the same as what would be returned by GPIB if ASYNCH were 0. Setting DIAG to a non-zero value causes a user defined function, IDLEFN, to be called within all the internal wait loops of the driver. This feature is used for diagnostic purposes and should not normally be used.

The assembly parameters in the utility file (IBUP.MAC) and the handler interface file (IBUDP.MAC) may also need editing. Table 4.4 lists the assembly parameters for them, their meanings, and their default values. The utility file also contains the device table that must be edited to describe the devices interfaced to the GPIB. A description of the information contained in the device table appears in Chapter 2 , Section 2.2.4.

After editing the driver and utility files, proceed to Step 4 for RSX handler installation, Step 5 for RT handler installation, and Step 6 for stand-alone installation or user interrupt routine installation under RT.

Table 4.4 - Driver Assembly Parameters

Name	Meaning	Default value 11-1	11V-1	11-2	11V-2
UADDR	UNIBUS (or Q-BUS) address of interface	167700	167700	167710	167710
VECT	Interrupt vector address RT legal values: 060-0474	300	300	310	310
PRI	Software interrupt priority as shown in the Processor Status Word. BR4 = 200, BR5 = 240, BR6 = 300, and BR7 = 340. The software priority must be equal to or greater than the hardware interrupt level.	200	200	200	200
ONLYC	Set to zero if more than one Controller is on GPIB. Set to non-zero value if interface is the only Controller on GPIB. If ONLYC is non-zero, code supporting pass control and bus monitoring is not assembled.	0	0	0	0
PPENAB	Parallel Poll Enable. If PPENAB is non-zero, code supporting the Parallel Poll function is assembled. If PPENAB is zero, the code is not assembled. On all interfaces except Rev D or later versions of the GPIB11-2, PPENAB must be zero if the hardware driver option switch is set to three-state mode. Otherwise, PPENAB must be non-zero if Parallel Polls are to be conducted.	N/A	N/A	1	1
TRI	High speed timing for three-state GPIB. If TRI is zero, open collector timing for handshake is used. If TRI is 4, high speed timing for handshake is used. In general, the TRI value must be consistent with the setting of the hardware driver option switch and with the value of PPENAB.	N/A	N/A	0	0

Table 4.4 - Driver Assembly Parameters (Continued)

Name	Meaning	Default value 11-1	11V-1	11-2	11V-2
SAC	System Controller. SAC must be non-zero if the interface card is the System Controller, and zero if the card is not the System Controller. SAC must agree with the setting of the hardware System Controller option switch where the switch exists.	N/A	N/A	1	1
EXT	Extended GPIB addressing. Set EXT to 2 if GPIB extended addressing is used and to 1 if normal addressing is used.	N/A	N/A	1	1
MSA	GPIB Secondary address. If EXT is 2, set MSA to an octal number representing the low five bits of the interface's GPIB secondary address. If EXT is 1, set MSA to 140.	N/A	N/A	140	140
MA	GPIB Primary address. Set MA to an octal number representing the low five bits of the interface's GPIB primary address.	N/A	N/A	25	25
The following parameters appear only in the stand-alone versions of the driver file:					
BASIC	BASIC support. Set BASIC to some non-zero number to enable assembly of code implementing the software interface to BASIC. Set BASIC to zero if you are implementing a FORTRAN/MACRO-callable driver.	0	0	0	0
OS	Set OS to 1 if the stand-alone driver is to be operated as a user interrupt service routine running under the RT operating system. If OS is 1, then code which notifies RT of interrupts is assembled. Do not use the stand-alone driver under RT unless you set OS to 1, otherwise, stack corruption and an eventual system crash will result.	0	0	0	0

Table 4.4 - Driver Assembly Parameters (Continued)

Name	Meaning	Default value	
		Utility File	Handler Interface File
BASIC	BASIC support. Set BASIC to a non-zero number if you are generating a BASIC-callable utility routine. Set BASIC to zero if you are generating a FORTRAN/MACRO-callable utility routine. If BASIC is non-zero, then code implementing the software interface to BASIC is assembled.	0	0
DVR	Driver installation Set DVR to 0 if you are generating a standalone version of the driver, or an RT user interrupt routine version of the driver. Set DVR to 1 if you are generating an RT handler version of the driver. Set DVR to 2 if you are generating an RSX handler version of the driver.	0	2

4.4 STEP 4: RSX INSTALLATION

The RSX handler version of the driver routine is supplied as a loadable driver. For more details on the installation of an RSX device driver refer to "RSX11-M Guide to Writing an I/O Driver" (AA-2600D-TC), Sections 3.3.2 through 3.3.5. The following procedure is essentially the same as the example for a loadable driver that is provided in the RSX manual.

Note 1: In this example of assembling and task-building the driver, the files RSXMC and RSX11M.STB must have been produced by the SYSGEN for the current system configuration. If a SYSGEN has been done on the system for a configuration other than the current configuration, files may exist with these names but the resulting installation of the driver will be incorrect.

Note 2: The RSX event flag is the value of the constant RSXEFN in the handler interface routine IBUDP. The RSX I/O channel is the value of the constant RSXCHAN in the same routine.

Note 3: The UIC's could vary depending on the version of RSX.

Assemble both the driver and its data base:

```
MAC>IBDRV,IBDRV=[1,1]EXEMC/ML,[200,200]RSXMC/PA:1,IBDRV
MAC>IBTAB,IBTAB=[1,1]EXEMC/ML,[200,200]RSXMC/PA:1,IBTAB
```

Then task-build the driver:

```
TKB>[1,54]IBDRV/-HD/-MM,,[1,54]IBDRV=
TKB>IBDRV,IBTAB
TKB>[1,54]RSX11M.STB/SS
TKB>[1,1]EXELIB/LB
TKB>/
TKB>STACK=0
TKB>PAR=DRVPAR:120000:10000
TKB>//
```

Note: The PAR option may specify any appropriate partition name but the address and size must be as shown, REGARDLESS OF THE ACTUAL PHYSICAL ADDRESS OF THE PARTITION!

Then load the driver:

```
LOAD IB:
```

Finally, assemble the handler interface and the utility routine:

```
MAC>IBUP,IBUP=IBUP
MAC>IBUDP,IBUDP=IBUDP
```

Proceed to Step 7 for BASIC installation and step 8 for FORTRAN/MACRO installation.

4.5 STEP 5: RT (TSX) HANDLER INSTALLATION

For more details on the installation of an RT device handler refer to "RT-11 System Generation Manual" (AA-5283B-TC), Section 3.6.5.

Note: sysgen RT with conditionalized timeout support.

Assemble the driver:

For RT Version 4

```
.R MACRO
*IB,IB=SYCND,IB
```

For RT Version 5

```
.R MACRO
*IB,IB=[Monitor], SYSGEN,CND.IB
```

where Monitor is substituted with XM.MAC, SJ.MAC, or FB.MAC

Then link the driver:

```
.R LINK
*IB.SYS=IB
```

Then install and load the driver:

```
.INSTALL IB:
.LOAD IB:
```

Finally, assemble the utility routine:

```
.R MACRO
*IBUP,IBUP=IBUP
*IBUDP,IBUDP=IBUDP
```

Proceed to Step 7 for BASIC installation and Step 8 for FORTRAN/MACRO installation.

4.6 STEP 6: STAND-ALONE INSTALLATION

The stand-alone version of the driver and utility routines can be assembled and linked on an RT system. RT-11 commands are shown in this step for purposes of illustration. The driver file is referred to as IBDP.MAC but should actually be the appropriate file corresponding to the GPIB interface being used. Assemble the driver file and the utility file:

```
.R MACRO
*IBDP,IBDP=IBDP
*IBUP,IBUP=IBUP
```

Proceed to Step 7 for BASIC installation and Step 8 for FORTRAN/MACRO installation.

4.7 STEP 7: BASIC INSTALLATION

For more details on the incorporation of assembly language routines into BASIC under RT, refer to "BASIC-11/RT-11 Installation Guide." If there are other assembly language routines to be linked in addition to these, then edit IBSCLI.MAC to include them in the function table. Assemble the function table:

```
MACRO/OBJECT:MYCLI BSMAC+BSASM+IBSCLI
```

Link BASIC together by running SUCNFG as described in the Installation Guide, being sure to answer the question "Do you want call support?" with "y." When prompted for module names, respond with MYCLI, IBDP, IBUP, and BSCLLB, all on separate lines, and then terminate the list with a blank line. Note that IBDP is NOT to be included in the list if the driver routine was installed as a device handler (Steps 4 or 5); in this case, specify IBUDP instead. If IBDP is included, it cannot be in an overlay region (since it must always be in memory to catch and service interrupts).

The resulting version of BASIC supports GPIB and IBUP operations as described in the manual, both in immediate mode and in program mode.

4.8 STEP 8: FORTRAN/MACRO INSTALLATION

The object modules IBDP.OBJ and IBUP.OBJ may be placed in the system library so that any references to them in a FORTRAN or MACRO program will be resolved automatically at link time. Note that IBDP is NOT to be placed in the library if the driver routine was installed as a device handler (steps 4 or 5). Instead, place IBUDP.OBJ in the system library if the driver was installed as a device handler. If IBDP is placed in the library, it can never be linked into an overlay region, since it must always be in memory to catch and service interrupts.

Section Five

INTERACTIVE CONTROL PROGRAM

IBIC is an interactive program that allows the GPIB to be controlled from terminal input using a convenient syntax. This program is especially useful for interactively debugging and trouble shooting a complete GPIB system - the bus, the instruments, the interface card, and the driver and utility software as well. IBIC may be used to quickly verify the behavior of an instrument instead of having to generate several versions of production software to do that. If a sequence of calls issued from IBIC works properly then it is reasonably certain (apart from possible timing considerations) that the same sequence of function calls issued from a program will work the same.

5.1 INSTALLATION

The IBIC source file must be edited to set the parameter OS to the proper value.

```
OS = 0 if stand-alone
OS = 1 if RT (handler or linkable routine)
OS = 2 if RSX
```

After being edited the source file is assembled with MACRO using the command:

```
IBIC,IBIC=IBIC
```

The object file is then linked or task-built with the driver and utility routines (see Chapter 4 for installation procedures for these modules) using the appropriate command for the target operating system:

IBIC=IBIC,IBUP,IBUDP	for TKB under RSX with driver installed as RSX handler
IBIC=IBIC,IBUP,IBUDP	for LINK under RT with the driver installed as an RT handler
IBIC=IBIC,IBUP,IBDP	for LINK under RT with the stand-alone driver, for use stand-alone or as user interrupt routine under RT

5.2 OPERATION

IBIC sends a prompt character (:) to the terminal and waits for input specifying the arguments to a GPIB or IBUP function call. The results of the function call are then printed on the terminal and another prompt is sent. The cycle repeats indefinitely until a control/c is typed (under RT) or a control/z (end-of-file) is typed (under RSX).

The syntax of the arguments is fairly simple and easily correlated with the FORTRAN and BASIC calling sequences as described in Chapters 2 and 3, and the semantics correspond exactly. The arguments are identical for all functions except COMMAND, WRITE, and READ. For COMMAND and WRITE, the buffer length is not specified (the string length is automatically counted by IBIC) and for READ an input buffer is not specified (one is supplied by IBIC).

Table 5.1 lists the format of the input to IBIC. All arguments enclosed in angle brackets (<>) stand for integers. All arguments enclosed in double quotes (") stand for strings. An integer is considered to be base 10 unless it begins with a 0 digit, in which case it is treated as base 8, e.g., 16 is the same as 020. A string is a sequence of characters that is enclosed in double quotes, e.g., "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, e.g., IBUP may be written out completely or abbreviated as I. The function code supplied to IBUP or GPIB may be the mnemonic shown in the table or the corresponding integer value, e.g.,

```
g cl
```

is equivalent to

```
g 4
```

Upper and lower case characters are treated the same for mnemonics but are distinct within a string. A special mechanism is 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 which specify the byte. For convenience a further abbreviation is available for the most common non-printing characters:

```
\r      stands for carriage return (equivalent to \15)
\n      stands for line feed (equivalent to \12)
\t      stands for tab (equivalent to \11)
\b      stands for backspace (equivalent to \10)
```

To actually include the backslash character in the string it is necessary to precede it with a backslash, e.g., \\ stands for the character \ by itself. A double quote may also be included in

the string by preceding it with a \ as in \".

The usual conventions apply for deleting a character or line from terminal input.

The returned values from the GPIB or IBUP calls are printed in decimal except for legal parallel poll, serial poll, and read command responses. These 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 above.

Table 5.1

IBIC INPUT FORMAT

```
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]

g[pib] co[mmand] "cmds"
g[pib] w[rite] "data" <wmd>
g[pib] rea[d] <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] m[onitor] <m>
g[pib] readc[ommand]
g[pib] setp[arameters] <t>
g[pib] te[stsrq] <w>
g[pib] f[inish]
g[pib] st[atus] <count>
g[pib] sp[byte] <byte>
```

5.3 IBIC EXAMPLE RUN

The following is an example sequence of GPIB function calls issued while running IBIC. While it illustrates the syntax of IBIC, it is not the only sequence that could be used to write to a device on the GPIB.

In this example the National Instruments interface is the System Controller (SAC).

IBIC Command	Explanation
: gpib finish 1	Unasserts all lines on the GPIB
: gpib setp 60 1	Sets the interface board's timeout value so the board will wait approximately 60 seconds for an operation to complete before returning a -6.
: gpib clear 1	Asserts IFC for 100 microseconds, then leaves IFC unasserted and ATN asserted.
: gpib remote 1	Asserts REN
: gpib command "_?" 2	Sends universal UNTalk and UNListen with ATN asserted and leaves ATN asserted.
: gpib command "U!" 2	Sends Talk Address octal 125 (ASCII U) Listen Address ocatal 41 (ASCII !) with last byte (write mode 0), leaving ATN unasserted.
: gpib write "\061\062\063" 0 3	Unasserts ATN and sends octal 61, 62, and 63 (ASCII 1,2,3), without asserting EOI with last byte (write mode 0), leaving ATN unasserted.
: gpib write "45678" 2 5	Unasserts ATN and sends octal 64, 65, 66, 67, and 70 (ASCII 4,5,6,7,8), asserts EOI with last byte (write mode 2), leaving ATN unasserted. Note: the above two gpib write function calls are split into two calls instead of one for the purpose of demonstration.
: gpib command "_?" 2	Sends universal UNTalk and UNListen with ATN asserted and leaves ATN asserted.

The following is an example sequence of ibup function calls issued while running IBIC. It illustrates the syntax of IBIC and is not the only sequence that could be used to write to a device on the GPIB. In this example, the National Instruments interface is the System Controller (SAC). Also, the device table in the file ibup.mac has been configured with slot 1 containing the appropriate information regarding GPIB addresses and the write mode. (Slot 0 is reserved for information about the interface.)

```
: ibup write 1 "12345678"
  8
```

Addresses the device defined in slot one of the device table, sends it the string of eight characters and unaddresses all device

If this is the first ibup function call issued (or the first one issued since a gpib finish or ibup finish) it would result in the same sequence of events as the gpib function cals above, starting with gpib clear, through the last gpib command "_?". If this was not the first ibup call issued, the results would be the same as the gpib function cals above, starting with the first gpib command "_?", through the last gpib command "_?". The one significant difference between the ibup call and the gpib sequence, while it would be unasserted after the ibup write call.

```
: ibup read 1 5
  5
  abcde
```

Addresses the device defined in slot one of the device table, reads from it a string of a maximum of five characters, prints out what it reads and unaddresses all devices.

```
: ibup define 1 0107 047 0 0 0 2
```

Redefines slot one in the device table. (This might be done to, say, change the read mode. The read mode of zero indicates that a read should terminate if EOI is sent.)

```
: ibup read 1 10
  5
  abcde
```

Addresses the device defined in slot one of the device table, reads from it a string of a maximum of ten characters, prints out what it reads and unaddresses all devices (In this case, since only five characters were read, the device talking must have asserted EoI with the fifth data byte to terminate the transmission.)

```
: ibup write 2 "big mess\r"
  -9
```

Addresses the device defined in slot 2 of the device table, tries to write to it and finds (by the state of the handshake lines) no one listening. (The problem could be that the address in the device table is incorrect or that device two is not hooked up to the GPIB.)

APPENDIX A

MULTILINE INTERFACE COMMAND MESSAGES
(Sent and Received with ATN TRUE)

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

APPENDIX A

MULTILINE INTERFACE COMMAND MESSAGES (CONT'D)
(Sent and Received with ATN TRUE)

Hex	Octal	Decimal	ASCII	Message	Hex	Octal	Decimal	ASCII	Message
40	100	64	@	MTA	60	140	96	`	MSA,PPE
41	101	65	A	MTA	61	141	97	a	MSA,PPE
42	102	66	B	MTA	62	142	98	b	MSA,PPE
43	103	67	C	MTA	63	143	99	c	MSA,PPE
44	104	68	D	MTA	64	144	100	d	MSA,PPE
45	105	69	E	MTA	65	145	101	e	MSA,PPE
46	106	70	F	MTA	66	146	102	f	MSA,PPE
47	107	71	G	MTA	67	147	103	g	MSA,PPE
48	110	72	H	MTA	68	150	104	h	MSA,PPE
49	111	73	I	MTA	69	151	105	i	MSA,PPE
4A	112	74	J	MTA	6A	152	106	j	MSA,PPE
4B	113	75	K	MTA	6B	153	107	k	MSA,PPE
4C	114	76	L	MTA	6C	154	108	l	MSA,PPE
4D	115	77	M	MTA	6D	155	109	m	MSA,PPE
4E	116	78	N	MTA	6E	156	110	n	MSA,PPE
4F	117	79	O	MTA	6F	157	111	o	MSA,PPE
50	120	80	P	MTA	70	160	112	p	MSA,PPD
51	121	81	Q	MTA	71	161	113	q	MSA,PPD
52	122	82	R	MTA	72	162	114	r	MSA,PPD
53	123	83	S	MTA	73	163	115	s	MSA,PPD
54	124	84	T	MTA	74	164	116	t	MSA,PPD
55	125	85	U	MTA	75	165	117	u	MSA,PPD
56	126	86	V	MTA	76	166	118	v	MSA,PPD
57	127	87	W	MTA	77	167	119	w	MSA,PPD
58	130	88	X	MTA	78	170	120	x	MSA,PPD
59	131	89	Y	MTA	79	171	121	y	MSA,PPD
5A	132	90	Z	MTA	7A	172	122	z	MSA,PPD
5B	133	91	[	MTA	7B	173	123	{	MSA,PPD
5C	134	92	\	MTA	7C	174	124	|	MSA,PPD
5D	135	93	]	MTA	7D	175	125	}	MSA,PPD
5E	136	94	^	MTA	7E	176	126	~	MSA,PPD
5F	137	95	_	UNT	7F	177	127	DEL	

APPENDIX B

RT HANDLER SPECIAL FUNCTION CODES

Special Function Code*	Driver Function Implemented	Description
-1	COMMAND	Similar to the WRITE I/O program request except that ATN is asserted while the data in the I/O buffer is written onto the GPIB.
-4	TRANSFER	Similar to the READ I/O program request except that the I/O buffer is unused.
-5	CLEAR	
-6	REMOTE	
-7	LOCAL	
-8	PARALLEL	The Parallel Poll byte is returned as the first byte in the I/O buffer.
-9	SERIAL	The status bytes received in the Serial Poll overwrite the command bytes originally in the I/O buffer.
-10	PASS CONTROL	
-11	SET STATUS	The "ist" bit is set if the I/O word count is non-zero and cleared if it is zero.
-12	MONITOR	Monitoring is enabled if the I/O word count is non-zero and disabled if it is zero.
-13	READ COMMAND	The next command byte is returned as the first byte in the I/O buffer. If no command bytes are available, an error is returned.
-14	-	This function is provided as a means to set the WRITE and READ modes and EOD. The first three bytes of the I/O buffer are used to set the WRITE mode, READ mode, and EOD character, respectively.
-15	TEST SRQ	An error is returned if SRQ is not asserted.
-16	FINISH	

*Decimal value unless otherwise noted.

APPENDIX C

RSX HANDLER QIO FUNCTION CODES

Special Function Code*	Driver Function Implemented	Description
0	-	Cancel I/O
1	WRITE	
2	READ	
3	-	Attach
4	-	Detach
5	COMMAND	Similar to WRITE except that ATN is asserted while the data bytes are written on the GPIB.
6	TRANSFER	Similar to READ except that the I/O buffer is not used.
7	CLEAR	This is a control function; no buffer or count is used.
8	REMOTE	This is a control function; no buffer or count is used.
9	LOCAL	This is a control function; no buffer or count is used.
10	PARALLEL	The Parallel Poll byte is returned in the second word of the I/O status block.
11	PASS CONTROL	This is a control function; no buffer or count is used.
12	SET STATUS	The "ist" bit is set if the first of the QIO device dependent parameters is non-zero and cleared if it is zero.

APPENDIX C

RSX HANDLER QIO FUNCTION CODES (CONT'D)

Special Function Code*	Driver Function Implemented	Description
13	MONITOR	Monitoring is enabled if the first of the QIO device dependent parameters is non-zero and disabled if it is zero.
14	READ COMMAND	The next monitored command byte is returned in the second word of the I/O status block. If no command bytes are available, an error is returned.
15	-	This function is provided as a means of setting the WRITE and READ modes and EOD. The first three bytes of the QIO device dependent parameters are used to set the WRITE mode, READ mode, and EOD character, respectively.
16	TEST SRQ	The first QIO device dependent parameter is a 1 if SRQ is to be waited for, or 0 if SRQ only is desired.
17	FINISH	This is a control function; no buffer or count is used.
18	STATUS	The buffer and count specify the location and amount of internal handler data to be returned.
19	SPBYTE	The first QIO device dependent parameter will become the byte next sent in response to a serial poll.

*Decimal value unless otherwise noted.

APPENDIX D

FORTRAN SYNTAX

UTILITY ROUTINE SUMMARY

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

DRIVER ROUTINE SUMMARY

Name	
COMMAND	J=GPIB(0,ARRAY,LENGTH)
WRITE	J=GPIB(1,ARRAY,LENGTH,MODE)
READ	J=GPIB(2,ARRAY,LENGTH,MODE,EOD)
TRANSFER	J=GPIB(3)
CLEAR	J=GPIB(4)
REMOTE	J=GPIB(5)
LOCAL	J=GPIB(6)
PARALLEL POLL	J=GPIB(7)
PASS CONTROL	J=GPIB(8)
SET STATUS	J=GPIB(9,S)
MONITOR	J=GPIB(10,M)
READ COMMAND	J=GPIB(11)
SET PARAMETERS	J=GPIB(12,T)
TEST SRQ	J=GPIB(13,W)
FINISH	J=GPIB(14)

BASIC SYNTAX

UTILITY ROUTINE SUMMARY

Name	
WRITE	CALL "IBUP"(0%,D%,W$)
READ	CALL "IBUP"(1%,D%,R%,C%)
CLEAR	CALL "IBUP"(2%,D%)
TRIGGER	CALL "IBUP"(3%,D%)
REMOTE	CALL "IBUP"(4%,D%)
LOCAL	CALL "IBUP"(5%,D%)
POLL	CALL "IBUP"(6%,D%,P%)
CONFIGURE	CALL "IBUP"(7%,D%,S%,L%)
PASS CONTROL	CALL "IBUP"(8%,D%)
DEFINE	CALL "IBUP"(9%,D%,T%,L%,S%,R%,E%,W%)
FINISH	CALL "IBUP"(10%)

DRIVER ROUTINE SUMMARY

Name	
COMMAND	CALL "GPIB"(0%,C$)
WRITE	CALL "GPIB"(1%,W$,M%)
READ	CALL "GPIB"(2%,R$,L%,M%,E%)
TRANSFER	CALL "GPIB"(3%)
CLEAR	CALL "GPIB"(4%)
REMOTE	CALL "GPIB"(5%)
LOCAL	CALL "GPIB"(6%)
PARALLEL POLL	CALL "GPIB"(7%,P%)
PASS CONTROL	CALL "GPIB"(8%)
SET STATUS	CALL "GPIB"(9%,S%)
MONITOR	CALL "GPIB"(10%,M%)
READ COMMAND	CALL "GPIB"(11%,C%)
SET PARAMETERS	CALL "GPIB"(12%,T%)
TEST SRQ	CALL "GPIB"(13%,W%,S%)
FINISH	CALL "GPIB"(14%)

BASIC+2 SYNTAX

UTILITY ROUTINE SUMMARY

Name	
WRITE	CALL IBUP BY REF(0%,D%,W$,LEN(W$),X%)
READ	CALL IBUP BY REF(1%,D%,R$,LEN(R$),X%)
CLEAR	CALL IBUP BY REF(2%,D%,X%)
TRIGGER	CALL IBUP BY REF(3%,D%,X%)
REMOTE	CALL IBUP BY REF(4%,D%,X%)
LOCAL	CALL IBUP BY REF(5%,D%,X%)
POLL	CALL IBUP BY REF(6%,D%,X%)
CONFIGURE	CALL IBUP BY REF(7%,D%,S%,L%,X%)
PASS CONTROL	CALL IBUP BY REF(8%,D%,X%)
DEFINE	CALL IBUP BY REF(9%,D%,T%,L%,S%,R%,E%,W%,X%)
FINISH	CALL IBUP BY REF(10%,X%)

DRIVER ROUTINE SUMMARY

Name	
COMMAND	CALL GPIB BY REF(0%,C$,LEN(C$),X%)
WRITE	CALL GPIB BY REF(1%,W$,LEN(W$),M%,X%)
READ	CALL GPIB BY REF(2%,R$,LEN(R$),M%,E%,X%)
TRANSFER	CALL GPIB BY REF(3%,X%)
CLEAR	CALL GPIB BY REF(4%,X%)
REMOTE	CALL GPIB BY REF(5%,X%)
LOCAL	CALL GPIB BY REF(6%,X%)
PARALLEL POLL	CALL GPIB BY REF(7%,X%)
PASS CONTROL	CALL GPIB BY REF(8%,X%)
SET STATUS	CALL GPIB BY REF(9%,S%,X%)
MONITOR	CALL GPIB BY REF(10%,M%,X%)
READ COMMAND	CALL GPIB BY REF(11%,X%)
SET PARAMETERS	CALL GPIB BY REF(12%,T%,X%)
TEST SRQ	CALL GPIB BY REF(13%,W%,X%)
FINISH	CALL GPIB BY REF(14%,X%)

After the BASIC+2 call the parameter X% will contain the value that would have been returned by the equivalent FORTRAN function call.

APPENDIX E

ERROR CODE SUMMARY

Mnemonic	FORTRAN/ MACRO Value*	Basic Message	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 "OR"ed with 0400 is returned. For READ COMMAND calls, the octal command byte "OR"ed with 0400 is returned.
OK	1	-	No error
-	0	-	Unused (special internal code)
-	-	alc--no space	No room to allocate string
ENONE	-1	-	No command byte available (READ COMMAND) or SRQ not asserted (TEST SRQ)
ECACFLT	-2	CAC conflict	ATN remains asserted after IFC sent (bus problem)
ENOTCAC	-3	not CAC	Not Active Controller for operation requiring CAC (software problem)
ENOTSAC	-4	not SAC	Not System Controller for operation requiring SAC (software problem)
EIFCLR	-5	IFC abort	IFC caused operation to abort (bus problem)
ETIMO	-6	timeout	Operation did not complete within allotted time (bus problem)
ENOFUN	-7	bad fctn code	Non-existent driver function code (software problem)
ETCTIMO	-8	TCT timeout	Take control not completed within allotted time (bus problem)
ENOIBDEV	-9	no listeners	No Listeners addressed or no devices connected (bus problem) or on GPIB11-2 GPIB11V-2 interfaces, a GPIB WRITE protocol error
EIDMACNT	-10	bcr error	Internal DMA completed without byte count going to zero (hardware problem)

ERROR CODE SUMMARY (CONT'D)

Mnemonic	FORTRAN/ MACRO Value*	Basic Message	Description
ENOPP	-11	no PP	PP operation attempted on three-state GPIB (software problem)
EITIMO	-12	ir timeout	Internal register DMA did not complete within allotted time (hardware problem)
EINEXM	-13	ir nex memory	Internal register DMA aborted due to non-existent memory (software/hardware problem)
ENEXMEM	-14	nex memory	GPIB DMA aborted due to non-existent memory (software/hardware problem)
ECNTRS	-15	bcr-bar error	Byte address and byte count are inconsistent following GPIB DMA (hardware problem)
ENOUMR	-16	UNIBUS map	no UNIBUS map register is available for the RSX handler (try again)
EOPEN	-17	open err	IB handler cannot be opened (software problem)
	-18	-	<unused>
-	-19	-	<unused>
ENOUFN	-20	bad ibup fctn code	Non-existent utility function code (software problem)
ENODEV	-21	no device	Illegal device slot number (software problem)
ENOLAD	-22	device not L	No listen address for selected device (software problem)
ENOTAD	-23	device not T	No talk address for selected device, or GPIB address in Define call is not legal (software problem)
EHDLR	-100	handler problem	communications problem with handler (probably incorrect installation)

*Decimal unless otherwise noted.

APPENDIX F

FIRST STEPS FOR INSTALLING A GPIB11-SERIES INTERFACE

STEP 1: WITH OPERATING SYSTEM NOT BOOTED AND INTERFACE BOARD REMOVED

Examine the addressable Q-BUS or UNIBUS registers of the interface board that are listed in Table F.1.

Expected Results: The processor halts or returns an error message on attempting to read non-existent I/O memory.

Other Results: If registers contents are printed out it indicates that some other device is already at the address intended for the GPIB interface. Pick another Q-BUS or UNIBUS base address and reconfigure the switches on the board (and likewise in the software).

STEP 2: WITH OPERATING SYSTEM NOT BOOTED AND INTERFACE BOARD INSTALLED

Double check Address, Vector, and GPIB Address switch settings on the board. With the power OFF, insert the card into the backplane. Power up the system. With the operating system still NOT booted, examine the addressable Q-BUS or UNIBUS registers listed in Table F.1.

Expected Results: See Table F.1.

Other Results: If processor halts or gives a non-existent memory error the Q-BUS or UNIBUS address switches on the board are not set correctly. Check carefully that all switches on the board reflect the desired addresses. If the contents are incorrect, another device may be set to the same Q-BUS or UNIBUS address, or the switches may not be set correctly on the board.

STEP 3: INSTALL THE SOFTWARE AND RUN IBIC

Make sure the software is installed properly (Chapter 4) and that the interface is configured as System Controller (SAC=1). Run the Interactive Control Program, IBIC (Chapter 5) to issue the following commands. (If the interface will not be used as the System Controller after this installation check,

reconfigure the software parameters and/or hardware switches appropriately after completing this step.)

STEP 3.1: USING IBIC, SEND GPIB CLEAR

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.

Other Results: To interpret error codes, see Appendix E. If system crashes or hangs, double check software parameters that were configured before assembling. Check EVERY FILE USED to make sure all assembly time parameters have been given a correct value. With GPIB11-2 boards check to see that the NPG wire is correctly configured on the backplane. On GPIB11-1 boards, check the VECTOR switch configuration on the board.

STEP 3.2: USING IBIC, SEND GPIB REMOTE

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

Other Results: To interpret error codes, see Appendix E.

STEP 3.3: USING IBIC, SEND GPIB COMMAND "?"

The ? in double quotes represents the UNListen command. This step requires that a Talker or Listener device that is known to work properly is connected to the interface to accept the commands. A bus tester 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 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, see Appendix E. If the system crashes or hangs, check the VECTOR switch settings on the board and the VECTOR value assigned in the software before assembly.

After completing these steps, use IBIC to issue any other commands that will be used in the application software. Again, a known working device must be attached to the GPIB to receive the commands. IMPORTANT: The sequence of commands is important for the proper operation of the GPIB. Consult the IEEE 488-1978 specification along with Chapters 2 and 3 of this manual for the correct protocol.

TABLE F.1

REGISTER CONTENTS AFTER BUS INIT WITH GPIB CABLE REMOVED

ADDRESS	REGISTER	CONTENTS
GPIB11-1		
base +0	RSR	000000
+2	RDB	000000
+4	TSR	000000 (+400 if SACS)
+6	TDB	000000
GPIB11V-1		
base +0	CTSR/ISR	000000* or 000020*
+2	CTSR/CSR	000000* or 000020*
+4	CTSR/ASR	000000*
+6	CTSR/ACR	000360*
+10	CTSR/ASWR	GPIB Address*(+40 if SACS) (+100 if EXT)
+12	CTSR/SPR	000000*
+14	CTSR/CPIR	000000*
+16	CTSR/DIR	000000* *(+400 if V8=1)
GPIB11-2 (through REV D)		
base +0	BCR	000000 [000000-177777]
+2	BAR	000000 [000000-177777]
+4	CSR	000200
+6	CCF	177777
GPIB11-2 (REV E)		
base +0	BCR	000000 [000000-177777]
+2	BAR	000000 [000000-177777]
+4	CSR	000200 (+2000 if OPEN COLLECTOR)
+6	CCF	177777
GPIB11V-2		
base +0	BCR	XXXXXX [000000-177777]
+2	BAR	XXXXXX [000000-177777]
+4	CSR	000277 (+2000 if OPEN COLLECTOR)
+6	XAR/CCF	140377 [140377-177777]

[] Indicates range of WRITE/READ registers. Values within this range can be deposited in the register and examined.